

# SMPTE STANDARD

## Material Exchange Format (MXF) – File Format Specification



Page 1 of 187

Table of Contents	Page
Foreword .....	7
Intellectual Property .....	7
<b>1 Scope.....</b>	<b>8</b>
<b>2 Conformance Notation .....</b>	<b>8</b>
<b>3 Normative References .....</b>	<b>8</b>
<b>4 Definition of Acronyms, Terms and Data Types .....</b>	<b>9</b>
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
<b>5 Introduction (Informative) .....</b>	<b>21</b>
5.1 Structure of this Document.....	22
5.2 SMPTE 377M Revisions.....	22
5.2.1 In-file version numbers .....	22
5.3 The MXF File .....	22
<b>6 Overall Specification.....</b>	<b>23</b>
6.1 Overall Data Structure .....	23
6.1.1 File Header .....	24
6.1.2 File Body .....	24
6.1.3 File Footer .....	24
6.2 Partitions.....	25
6.2.1 Partition Rules Overview .....	25
6.2.2 Partition Rules Summary (Informative) .....	27
6.2.3 Partition Status.....	27
6.2.4 The Status of an MXF File .....	28
6.2.5 Header Partition.....	29
6.2.6 Body Partition .....	29
6.2.7 Footer Partition .....	29
6.2.8 Using Partitions to multiplex Essence Containers and associated Index Tables (Informative) .....	29
6.3 KLV Coding.....	30
6.3.1 KLV Coding Sequence .....	30
6.3.2 KLV Coded Dark Components .....	31
6.3.3 KLV Fill Items.....	31
6.3.4 KLV Lengths .....	32
6.3.5 Local set Lengths.....	32

6.3.6	Variable-Length Pack Lengths .....	32
6.3.7	Defined-Length Pack Lengths .....	32
6.3.8	MXF Keys and Universal Labels .....	33
6.3.9	Constraints on recursive Groupings of KLV items .....	33
6.3.10	The Primer Pack, Dark Metadata and MXF extensions .....	33
<b>6.4</b>	<b>MXF Encoding Requirements .....</b>	<b>34</b>
6.4.1	KLV Alignment Grid (KAG) .....	35
6.4.2	MXF Byte Order .....	36
6.4.3	Encoding Constraints .....	36
<b>6.5</b>	<b>Manipulating files conforming to other revisions of this specification .....</b>	<b>36</b>
<b>6.6</b>	<b>Run-In Sequence .....</b>	<b>36</b>
<b>6.7</b>	<b>Minimum MXF Decoder (Informative) .....</b>	<b>37</b>
<b>6.8</b>	<b>Strong and Weak Reference Integrity (Informative) .....</b>	<b>37</b>
<b>7</b>	<b>Partitions .....</b>	<b>38</b>
<b>7.1</b>	<b>Partition Pack .....</b>	<b>38</b>
<b>7.2</b>	<b>Header Partition Pack .....</b>	<b>41</b>
7.2.1	Header Partition Pack Key .....	41
7.2.2	Header Partition Pack Values .....	42
<b>7.3</b>	<b>Body Partition Pack .....</b>	<b>42</b>
7.3.1	Body Partition Pack Key .....	42
7.3.2	Body Partition Pack Value .....	43
7.3.3	Header Metadata Repetition in Body Partitions .....	43
<b>7.4</b>	<b>Footer Partition Pack .....</b>	<b>43</b>
7.4.1	Footer Partition Pack Key .....	43
7.4.2	Footer Partition Pack Value .....	44
7.4.3	Header Metadata Repetition in the Footer Partition .....	44
<b>7.5</b>	<b>Header Metadata Repetition in Body and Footer Partitions .....</b>	<b>44</b>
7.5.1	Application Guidelines for header Metadata Repetition (Informative) .....	44
7.5.2	Tracking Changes with Generation UID .....	45
<b>8</b>	<b>Operational Patterns .....</b>	<b>46</b>
<b>8.1</b>	<b>General .....</b>	<b>46</b>
<b>8.2</b>	<b>Generic Universal Label for All Operational Patterns .....</b>	<b>46</b>
<b>8.3</b>	<b>Generalized Operational Patterns .....</b>	<b>47</b>
8.3.1	Item complexity .....	47
8.3.2	Package complexity .....	47
8.3.3	Universal Label for Generalized Operational Patterns .....	48
<b>8.4</b>	<b>Specialized Operational Patterns .....</b>	<b>49</b>
8.4.1	Universal Label byte values for Specialized Operational Patterns .....	49
8.5	Package Hierarchy in Operational Patterns .....	50
<b>9</b>	<b>Header Metadata .....</b>	<b>50</b>
<b>9.1</b>	<b>Header Metadata KLV Packet Sequence .....</b>	<b>51</b>
<b>9.2</b>	<b>Primer Pack .....</b>	<b>51</b>
9.2.1	Contents of the Primer .....	53
9.2.2	Local Tag values .....	53
9.2.3	Dark Metadata Support .....	53
<b>9.3</b>	<b>Header Metadata Set Coding .....</b>	<b>54</b>
9.3.1	Data Model (Informative) .....	55
9.3.2	Strong and Weak References .....	55
9.3.3	Uniqueness of Instance UID values .....	56
<b>9.4</b>	<b>Structural Metadata Semantics .....</b>	<b>56</b>
9.4.1	Explanation of Figures illustrating the Structural Metadata Semantics (Informative) .....	56
9.4.2	The MXF timing Model .....	58
9.4.3	Relationship between File Packages and Essence Containers .....	61
<b>9.5</b>	<b>Structural Metadata Definition .....</b>	<b>61</b>

9.5.1	Header Metadata start.....	61
9.5.2	Generic Class diagram (Informative).....	62
9.5.3	Material Package .....	64
9.5.4	Source Package .....	64
9.5.5	Top-Level File Packages .....	64
9.5.6	Lower-Level Source Packages.....	66
9.5.7	Relationship between the Packages and SourcePackageID / SourceTrackID .....	66
9.5.8	Relationship between the BodySID and IndexSID .....	68
9.5.9	Scope of the Track ID values .....	68
<b>9.6</b>	<b>Structural Header Metadata Implementation .....</b>	<b>68</b>
9.6.1	KLV Key values for Structural Metadata Sets .....	68
9.6.2	Universal Labels for Abstract Structural Metadata Groups .....	70
<b>9.7</b>	<b>Application Metadata Plug-Ins.....</b>	<b>72</b>
9.7.1	General (Informative).....	72
9.7.2	Application Metadata Scheme Specification.....	72
9.7.3	Generic Universal Label for the MXF Application Metadata Schemes.....	72
9.7.4	Plug-In Mechanism.....	73
9.7.5	Simple Application Metadata Plug-In Instance Removability.....	75
9.7.6	Simple Application Metadata Plug-In Instance Removal Implementation (Informative).....	76
9.7.7	Use of the Application Metadata Plug-In Mechanism.....	76
9.7.8	Application-Specific Metadata Plug-In Mechanism Example (Informative) .....	76
<b>9.8</b>	<b>Descriptive Metadata Plug-Ins.....</b>	<b>79</b>
9.8.1	General (Informative).....	79
9.8.2	Generic Universal Label for the MXF Descriptive Metadata Schemes .....	79
9.8.3	Generic MXF Descriptive Metadata Keys.....	80
9.8.4	Universal Labels for Abstract Descriptive Metadata Groups .....	81
9.8.5	Plug-In Mechanism.....	82
9.8.6	Simple DM Plug-In Instance Removability.....	84
9.8.7	Simple DM Plug-In Instance Removal Implementation (Informative).....	85
<b>9.9</b>	<b>Start Timecode Values .....</b>	<b>85</b>
<b>10</b>	<b>File Body .....</b>	<b>86</b>
<b>10.1</b>	<b>Essence Containers.....</b>	<b>86</b>
<b>10.2</b>	<b>Technical Requirements for MXF Essence Containers .....</b>	<b>86</b>
<b>10.3</b>	<b>Standards Requirements of an MXF Essence Container document .....</b>	<b>87</b>
<b>10.4</b>	<b>General Information (Informative).....</b>	<b>87</b>
<b>10.5</b>	<b>Descriptors .....</b>	<b>87</b>
10.5.1	Use of Descriptors in File Packages.....	88
10.5.2	Use of Descriptors in Physical Packages .....	88
10.5.3	Use of Locators .....	89
10.5.4	Extending Essence Descriptors .....	89
<b>10.6</b>	<b>Interleaved Essence Containers.....</b>	<b>89</b>
<b>11</b>	<b>Index Table .....</b>	<b>90</b>
<b>11.1</b>	<b>Overview .....</b>	<b>90</b>
11.1.1	Interleaved Streams .....	91
11.1.2	Constant Bytes per Element (CBE) and Variable Bytes per Element (VBE) (Informative).....	91
11.1.3	Complex Interleaves of Compressed Audio .....	92
11.1.4	Description of Operation .....	93
11.1.5	Generalization using Element Date .....	94
11.1.6	Temporal Re-ordering .....	95
11.1.7	Indexing Empty Essence Elements.....	96
11.1.8	Indexing KLV Fill Items.....	96
11.1.9	Constant Edit Unit Size.....	96
<b>11.2</b>	<b>Index Table Specification.....</b>	<b>97</b>
11.2.1	Index Table Segments.....	98
11.2.2	Index Table Segment Key .....	98

11.2.3	Index Table Segment .....	100
<b>11.3</b>	<b>Partial / Sparse Index Tables for VBE Essence .....</b>	<b>105</b>
<b>11.4</b>	<b>To Find the Byte Offset for an Essence Element (Informative) .....</b>	<b>106</b>
<b>11.5</b>	<b>Using Index Tables for Internal Essence and External Essence .....</b>	<b>106</b>
11.5.1	BodySID nonzero, IndexSID nonzero .....	106
11.5.2	BodySID zero, IndexSID nonzero .....	106
11.5.3	BodySID nonzero, IndexSID zero .....	107
11.5.4	BodySID zero, IndexSID zero .....	107
<b>11.6</b>	<b>Additional Information (Informative) .....</b>	<b>107</b>
11.6.1	Relationship between Top-Level File Package Essence Timeline Tracks and Index Entries .....	107
11.6.2	Look-up Algorithm for Conversion of Index Position to Stream Offset .....	107
<b>12</b>	<b>Random Index Pack .....</b>	<b>109</b>
<b>12.1</b>	<b>Random Index Pack Key .....</b>	<b>109</b>
<b>12.2</b>	<b>The Random Index Pack Value .....</b>	<b>110</b>
<b>12.3</b>	<b>Algorithm for using the Random Index Pack (Informative) .....</b>	<b>110</b>
<b>Annex A</b>	<b>Specifications for Root Metadata Sets (normative) .....</b>	<b>111</b>
<b>A.1</b>	<b>Interchange Object .....</b>	<b>111</b>
<b>A.2</b>	<b>Preface .....</b>	<b>112</b>
<b>A.3</b>	<b>Identification .....</b>	<b>114</b>
<b>A.4</b>	<b>Content Storage .....</b>	<b>115</b>
<b>A.5</b>	<b>Essence Container Data .....</b>	<b>116</b>
<b>Annex B</b>	<b>Specifications for the Generic Package (normative) .....</b>	<b>118</b>
<b>B.1</b>	<b>Generic Package .....</b>	<b>118</b>
<b>B.2</b>	<b>Generic Descriptor .....</b>	<b>118</b>
<b>B.3</b>	<b>SubDescriptor .....</b>	<b>119</b>
<b>B.4</b>	<b>Network Locator .....</b>	<b>119</b>
B.4.1	URL file:// .....	119
B.4.2	URL ftp:// .....	120
B.4.3	URIs .....	120
<b>B.4.4</b>	<b>Handling invalid or unknown URLs and URIs .....</b>	<b>120</b>
<b>B.5</b>	<b>Text Locator .....</b>	<b>120</b>
<b>B.6</b>	<b>Generic Track .....</b>	<b>120</b>
<b>B.7</b>	<b>Track ID Usage .....</b>	<b>121</b>
<b>B.8</b>	<b>Structural Component .....</b>	<b>122</b>
<b>B.9</b>	<b>Sequence .....</b>	<b>122</b>
<b>B.10</b>	<b>Source Clip .....</b>	<b>123</b>
<b>B.11</b>	<b>Filler .....</b>	<b>123</b>
<b>B.12</b>	<b>Timeline Track .....</b>	<b>124</b>
<b>B.13</b>	<b>Track Event .....</b>	<b>124</b>
<b>B.14</b>	<b>Static Track .....</b>	<b>125</b>
<b>B.15</b>	<b>Timeline Track (Timecode) .....</b>	<b>125</b>
<b>B.16</b>	<b>Sequence (Timecode) .....</b>	<b>126</b>
<b>B.17</b>	<b>Timecode Component .....</b>	<b>126</b>
<b>B.18</b>	<b>Timeline Track (Picture) .....</b>	<b>127</b>
<b>B.19</b>	<b>Sequence (Picture) .....</b>	<b>127</b>
<b>B.20</b>	<b>Source Clip (Picture) .....</b>	<b>127</b>
<b>B.21</b>	<b>Timeline Track (Sound) .....</b>	<b>128</b>
<b>B.22</b>	<b>Sequence (Sound) .....</b>	<b>128</b>
<b>B.23</b>	<b>Source Clip (Sound) .....</b>	<b>128</b>
<b>B.24</b>	<b>Timeline Track (Data) .....</b>	<b>129</b>
<b>B.25</b>	<b>Sequence (Data) .....</b>	<b>130</b>
<b>B.26</b>	<b>Source Clip (Data) .....</b>	<b>130</b>
<b>B.27</b>	<b>Dm Tracks .....</b>	<b>130</b>
B.27.1	Timeline Track (DM) .....	130

B.27.2 Event Track (DM) .....	131
B.27.3 Static Track (DM) .....	131
<b>B.28 Sequence (DM) .....</b>	<b>131</b>
<b>B.29 Segment .....</b>	<b>131</b>
<b>B.30 Event .....</b>	<b>132</b>
<b>B.31 Comment Marker .....</b>	<b>132</b>
<b>B.32 DM Segment .....</b>	<b>133</b>
<b>B.33 DM Source Clip .....</b>	<b>134</b>
<b>B.34 Package Marker Object .....</b>	<b>134</b>
<b>Annex C Specification of the Application-Specific Metadata Plug-In Mechanism Sets (normative)</b> .....	<b>136</b>
<b>C.1 Application Object .....</b>	<b>136</b>
<b>C.2 Application Plug-In Object .....</b>	<b>136</b>
<b>C.3 Application Referenced Object .....</b>	<b>138</b>
<b>Annex D Specification of the DM Plug-In Mechanism Sets (normative) .....</b>	<b>139</b>
<b>D.1 Descriptive Framework .....</b>	<b>139</b>
<b>D.2 Descriptive Object .....</b>	<b>139</b>
<b>Annex E Specification for the Packages used in MXF (normative) .....</b>	<b>140</b>
<b>E.1 Material Package .....</b>	<b>140</b>
<b>E.2 Source Package .....</b>	<b>140</b>
<b>E.3 File Package .....</b>	<b>141</b>
<b>E.4 Physical Package .....</b>	<b>141</b>
<b>E.5 Package hierarchy in MXF .....</b>	<b>141</b>
<b>Annex F Specification of Descriptors used in MXF (normative) .....</b>	<b>143</b>
<b>F.1 Scope of Descriptor Property Values .....</b>	<b>143</b>
<b>F.2 File Descriptor .....</b>	<b>143</b>
<b>F.3 Multiple Descriptor .....</b>	<b>145</b>
<b>F.4 Picture Essence Descriptors .....</b>	<b>145</b>
F.4.1 Generic Picture Essence Descriptor .....	145
F.4.2 CDCI (Color Difference Component Image) Picture Essence Descriptor .....	149
<b>F.4.3 RGBA (Red Green Blue Alpha) Picture Essence Descriptor .....</b>	<b>151</b>
<b>F.5 Generic Sound Essence Descriptor .....</b>	<b>152</b>
<b>F.6 Generic Data Essence Descriptor .....</b>	<b>153</b>
<b>Annex G Picture Essence Descriptor Properties (normative) .....</b>	<b>154</b>
<b>G.1 Data Storage, Sampling, Display and Video Interface .....</b>	<b>154</b>
G.1.1 Stored Data and Stored Rectangle .....	155
G.1.2 Sampled Rectangle .....	156
G.1.3 Display Rectangle .....	156
G.1.4 Video Interface .....	156
G.1.5 Sampling .....	157
<b>G.2 Property Definitions .....</b>	<b>157</b>
G.2.1 Frame Layout .....	157
G.2.2 Sample Rate and Edit Rate .....	159
G.2.3 Signal Standard .....	159
G.2.4 Aspect Ratio .....	159
G.2.5 Active Format Descriptor (AFD) .....	160
G.2.6 Stored Width .....	161
G.2.7 Stored Height .....	161
G.2.8 Sampled Width .....	162
G.2.10 SampledXOffset .....	162
G.2.11 SampledYOffset .....	162
G.2.12 Video Line Map .....	163
G.2.13 DisplayWidth .....	163

G.2.14	DisplayHeight .....	163
G.2.15	DisplayXOffset .....	164
G.2.16	DisplayYOffset .....	164
G.2.17	DisplayF2Offset .....	164
G.2.18	StoredF2Offset .....	164
G.2.19	FieldDominance .....	165
G.2.20	Alpha Transparency .....	165
G.2.21	Transfer Characteristic .....	165
G.2.22	Image Alignment Offset .....	166
G.2.23	Image Start Offset .....	166
G.2.24	Image End Offset .....	166
G.2.25	Picture Essence Coding .....	166
G.2.26	Component Depth .....	170
G.2.27	Horizontal Subsampling .....	170
G.2.28	Vertical Subsampling .....	170
G.2.29	Color Siting .....	170
G.2.30	PaddingBits .....	171
G.2.31	Alpha Sample Depth .....	171
G.2.32	Black Ref Level .....	171
G.2.33	White Ref Level .....	172
G.2.34	Color Range .....	172
G.2.35	Reversed Byte Order .....	172
G.2.36	PixelLayout .....	172
G.2.37	Palette .....	173
G.2.38	PaletteLayout .....	173
G.2.39	Scanning Direction .....	173
G.2.40	Pixel Layout .....	174
<b>Annex H Static Local Tags Assigned by MXF Specifications (normative) .....</b>		<b>176</b>
<b>Bibliography .....</b>		<b>185</b>

## 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 its Standards Operations Manual.

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

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 335:2012, Television — Metadata Dictionary Structure



SMPTE ST 336:2017, Data Encoding Protocol Using Key-Length-Value

SMPTE ST 395:2014, Television — Metadata Groups Registry Structure

SMPTE ST 400:2012, Television — SMPTE Labels Structure

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

IEEE 754-2019, Floating Point Format

ITU-R BS.1196-7 (10/2015), Audio Coding for Digital Broadcasting

IETF RFC 3986 (2005), Uniform Resource Identifier (URI): Generic Syntax

IETF RFC 4122 (2005), A Universally Unique Identifier (UUID) URN Namespace

ISO/IEC 8825-1:2015 Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

ISO/IEC 10646:2017, 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 constraint 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.

**RP 210 definition:** text in a note copied from the SMPTE Dictionary RP 210 circa 2004 and updated to prevent divergence with the SMPTE Metadata registers in 2019. The **RP210 definition** term is used in Informative Notes within tables in this document. The superseded RP 210 was the original 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 superseded RP210 and the SMPTE Metadata Registers.

**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 below. 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.34.01.01.01.07.02.01.01.01.04.00.00 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 <sup>th</sup> 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 <sup>th</sup> 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 <sup>th</sup> 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

The following is an overview of in-file version numbers for SMPTE 377-1.

### 5.2.1 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

## 5.3 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.

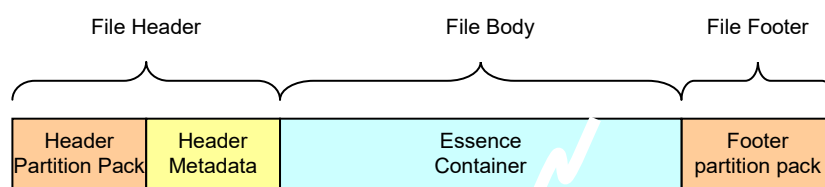
## 6 Overall Specification

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

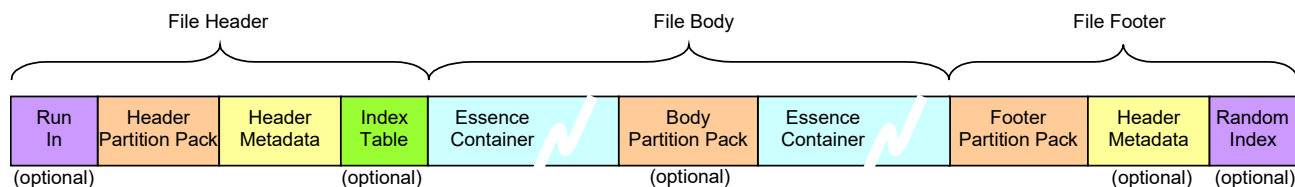
### 6.1 Overall Data Structure

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 called 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.

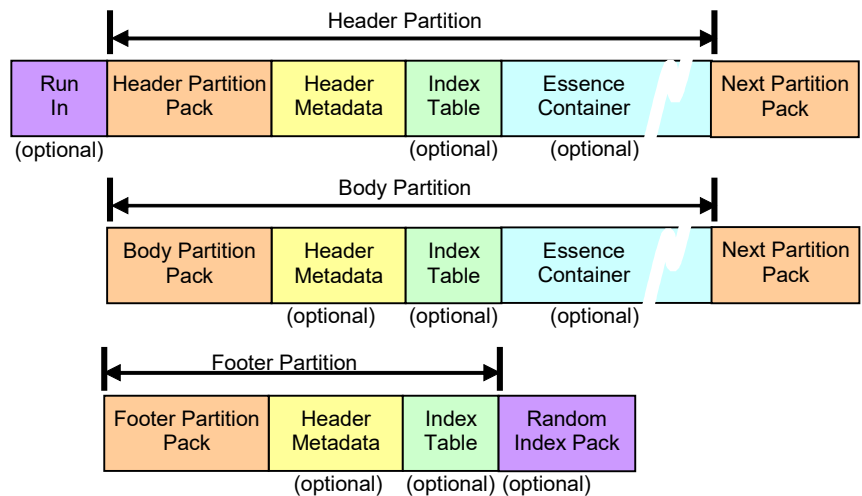


Figure 3 – Required order of file components in each Partition kind

### 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.

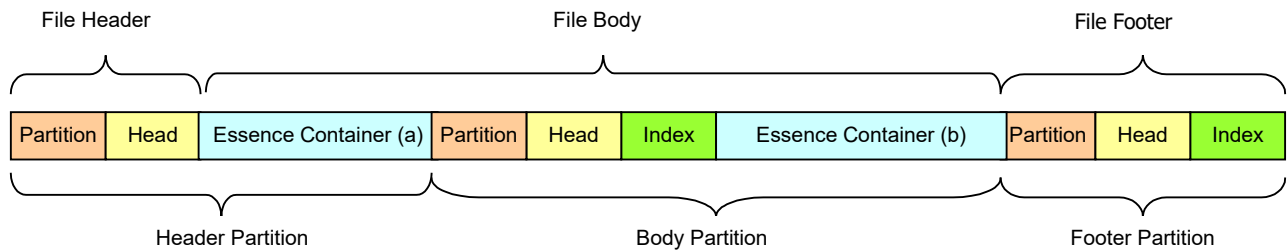


Figure 4 – MXF File Containing One Essence Container

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.

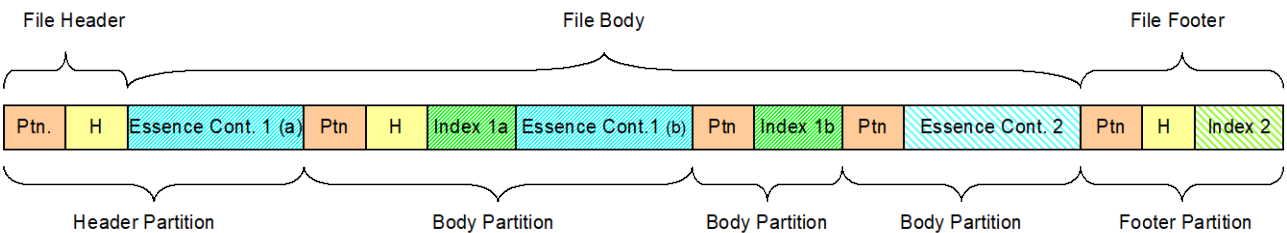


Figure 5 – MXF File Containing Two Essence Containers

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.

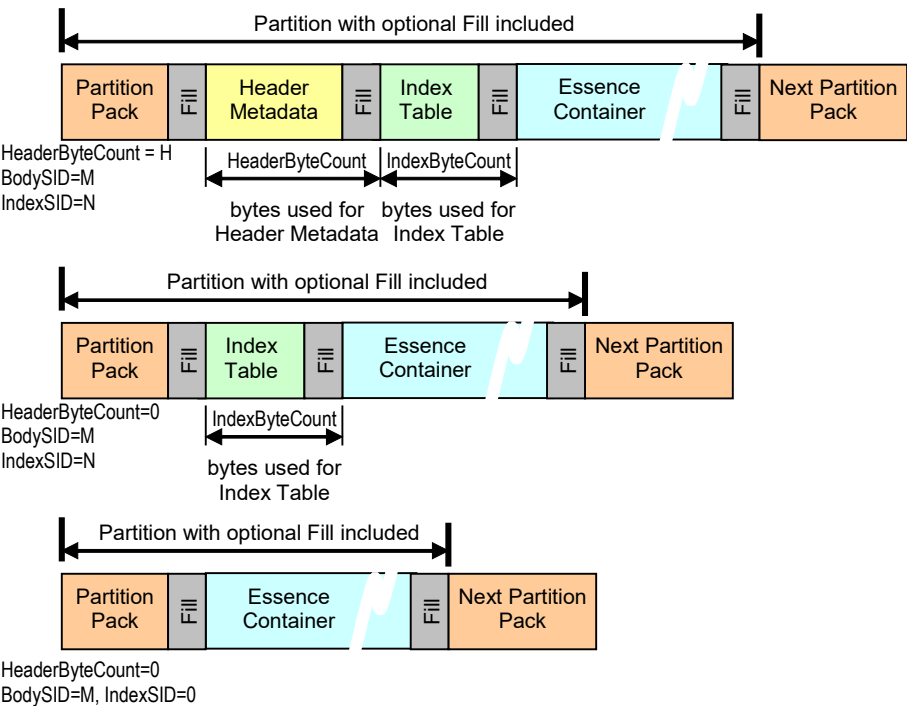


Figure 6 – Ordering in a Partition and the use of Fill Items

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 Manipulating files conforming to other revisions of this specification**

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.

### **6.6 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.

## 6.7 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).

## 6.8 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. [Note: SMPTE RP 210 definition: 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. [Note: SMPTE RP 210 definition: 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 [Note: SMPTE RP 210 definition: 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 <b>byte offset</b> relative to the start of the Header Partition). [Note: SMPTE RP 210 definition: 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 <b>byte offset</b> relative to the start of the Header Partition). [Note: SMPTE RP 210 definition: 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 <b>byte offset</b> relative to the start of the Header Partition). [Note: SMPTE RP 210 definition: 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. [Note: SMPTE RP 210 definition: 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	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	0

Item Name	Type	Len	Item UL	Meaning	Default
			00.00.00.00	KLV Fill item which is included in the IndexByteCount. See Figure 6. [Note: SMPTE RP 210 definition: 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. [Note: SMPTE RP 210 definition: 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 [Note: SMPTE RP 210 definition: 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. [Note: SMPTE RP 210 definition: 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) [Note: SMPTE RP 210 definition 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) [Note: SMPTE RP 210 definition: 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 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 or zero (0). If the Footer Partition is not present in the file then this Property shall be zero (0).

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 UUIDs 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.

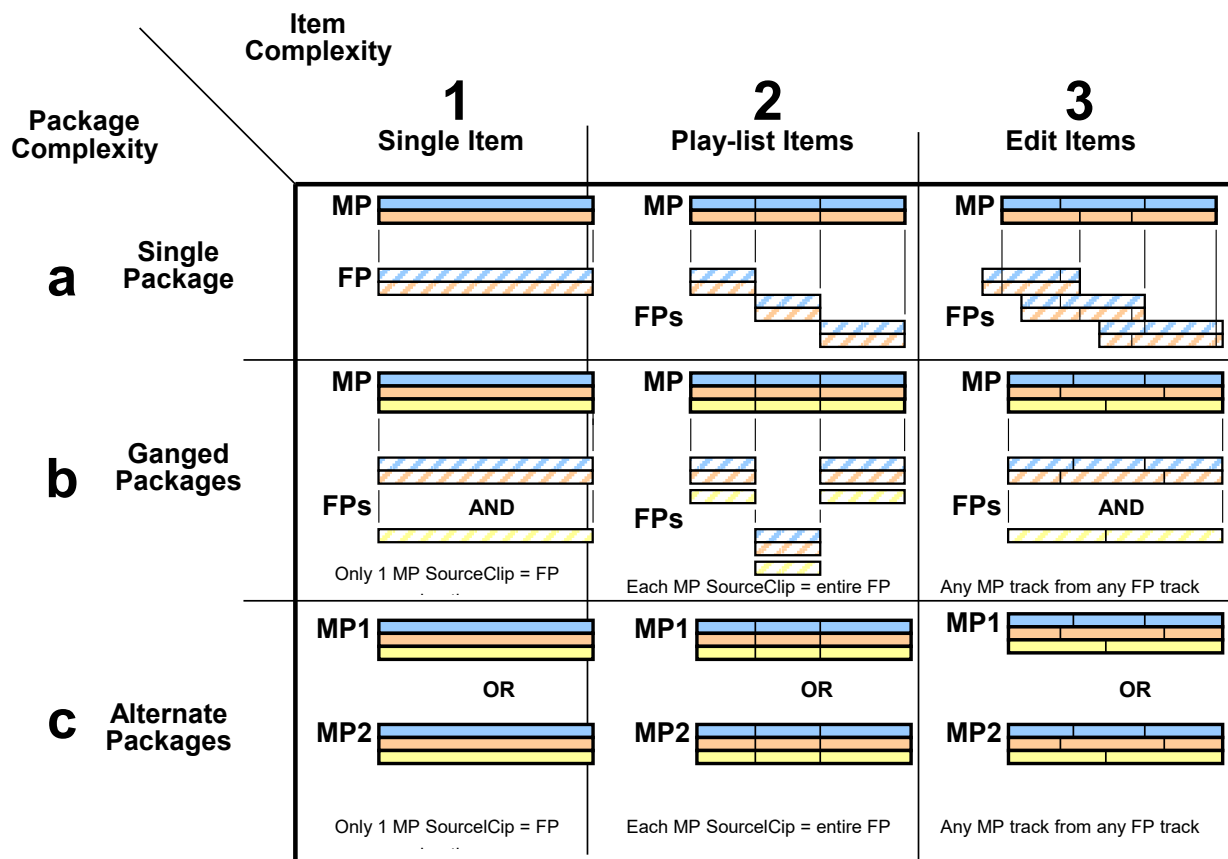


Figure 7 – Illustration of the Operational Pattern Axes (Informative)

In Generalized Operational Patterns, the Material Package shall reference one or more Top-Level File Packages.

For Generalized Operational Patterns, MXF encoders shall identify the correct values of the Item and Package Complexity through the Operational Pattern UL in all Partition Packs.

Applications creating files conforming to Generalized Operational Patterns shall signal the simplest Operational Pattern to which a file conforms (i.e. if a file conforms to the constraints of OP1b, it shall not be identified as OP2b or OP1c).

8.3.3 Universal Label for Generalized Operational Patterns

For Generalized Operational Patterns, certain axes of flexibility have been defined and the normative value ranges are tabulated here. For specific values of bytes 13 and 14, the associated Operational Pattern document should be consulted.



**Table 10 – Universal Label byte ranges for Generalized Operational Patterns**

Byte No.	Description	Value (hex)	Meaning
13	Operational Pattern Definition	01h–03h	Item Complexity 01h = Single Item 02h = Play-list Items 03h = Edit items
14	Operational Pattern Definition	01h–03h	Package Complexity 01h = Single Package 02h = Ganged Packages 03h = Alternate Packages
15	Operational Pattern Definition	zzh	Qualifier bits see Table 11
16	Operational Pattern Definition	nnh	Reserved for specification in OP documents

The values of byte 15 shall be defined according to the entries in the table below. This byte qualifies the Operational Pattern in a way which shall apply to all Generalized Operational Patterns.

**Table 11 – Byte 15 values of the Generalized MXF Operational Pattern UL**

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

## 8.4 Specialized Operational Patterns

For Specialized Operational Patterns, it is likely that very detailed and specialized constraints will apply. These shall be defined in the associated Specialized Operational Pattern specification.

### 8.4.1 Universal Label byte values for Specialized Operational Patterns

Values for byte 13 of the Operational Pattern UL in the range '10h' to '7Fh' are reserved to allow Specialized Operational Patterns to be uniquely identified. The meaning of the final 3 bytes of the label shall be defined in the specification of the Specialized Operational Pattern.

**Table 12 – Byte ranges of the Specialized MXF Operational Pattern UL**

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

## 8.5 Package Hierarchy in Operational Patterns

Where present, the Primary Package Property of the Preface Set identifies the Package which an MXF decoder shall treat as the default Package. The default Package shall be the Package that, in the absence of external control information is played by an MXF decoder.

Example: Primary Package could identify which Material Package, of several in an MXF file, shall be played by default.

The particular specialized use of Primary Package may be defined by specialized Operational Pattern specifications.

The difference between Package types is detailed in Annex E.5.

## 9 Header Metadata

The Header Metadata contains Metadata Sets which defines the contents of the file as a whole, including any Essence Containers. It may also describe Essence that is stored external of the file.

The Header Metadata is broadly split into two categories: Structural Metadata (see Sections 9.5 and 9.6) and Descriptive Metadata (see Section 9.8).

Note: Application-Specific Metadata (see Section 9.7) can be used to extend either of the two categories. However, extending Descriptive Metadata is not recommended unless the application that defines the Application Scheme is the same as the application that defines the Descriptive Metadata Scheme.

All three categories shall be encoded as a single sequence of KLV coded packets.

This specification defines Structural Metadata as a single extensible scheme. There shall be no other Structural Metadata schemes in MXF, i.e. there shall be no other scheme of similar or identical functionality that replaces the scheme that is defined in Section 9.5.

Within the Header Metadata, any Descriptive Metadata is defined as a 'plug-in'. MXF files may contain instances of one or more Descriptive Metadata Schemes. A Descriptive Metadata Scheme shall either be an MXF Descriptive Metadata Scheme that is identified by a Label defined in Section 9.8.2 or a non-MXF Descriptive Metadata Scheme that is identified by another UL. All such Descriptive Metadata Schemes shall use the plug-in mechanism defined in Section 9.8.

Within the Header Metadata, any Application-Specific Metadata is defined as a 'plug-in'. MXF files may contain instances of one or more Application Metadata Schemes. All Application Metadata Schemes shall use the plug-in mechanism defined in Section 9.7.

Within the Header Metadata, extensions may be defined by specifying Subclasses, by 'plug-ins' as specified by Section 9.7, or by the addition of Optional Properties to existing MXF Sets as described in Section 9.3.

Note 1: An example for the use of Subclasses is the definition of extensions to existing Generic Descriptors by new MXF Essence Mapping documents (see Section 10.5.4).

Note 2: The 'plug-in' mechanism specified in Section 9.7 defines one possible way to encode extensions that use Subclasses such that the extensions are transparent to basic MXF applications that do not recognize them (see Section 9.7.8 for an example), but that implement Superclass from which the Subclasses are derived. In addition to the direct KLV encoding of Subclasses that are identified by SMPTE ULs, other SMPTE Engineering Documents can define other encoding mechanisms, especially for Subclasses that are identified by UUIDs.

## 9.1 Header Metadata KLV Packet Sequence

The Header Metadata shall start with the Primer Pack.

There may be one KLV Fill item that separates the Partition Pack and the Primer Pack. No other KLV packet shall be permitted between the Partition Pack and the Primer Pack.

There may be one KLV Fill item that separates the Primer Pack and first Header Metadata Set.

KLV packets may be inserted between the Primer Pack and the first Header Metadata Set. Unless their KLV keys identify them as Class 14 (i.e. privately registered for private use), their semantics and application shall be defined by a published SMPTE Engineering Document. The total byte count of these KLV encoded dark components shall be included in the value of HeaderByteCount of the Partition Pack.

Note 1: While there is no guarantee that baseline MXF applications will preserve this data or will maintain its position within the file, application specifications can require that the data and its position are preserved.

Note 2: As defined in Section 9.5.1, the first Header Metadata Set is the Preface Set.

The Header Metadata may be followed by a single KLV Fill Item. There may be KLV Fill Items anywhere within the Header Metadata.

Note 3: As defined in Section 7.1, the KLV Fill item following the Header Metadata is included in the value of HeaderByteCount of the Partition Pack.

## 9.2 Primer Pack

The Primer Pack is a look up table which ensures that all 2-byte Local Tags in all Header Metadata Sets are unique within a Partition.

Note: This is a constraint on the scope rules of SMPTE ST 336.

The Primer Pack provides a mapping from all 2-byte Local Tags to their respective UIDs for all MXF defined Sets; i.e. all those defined in any of the documents in the MXF specification.

The scope of the Local Tags in the Primer Pack shall be restricted to the Header Metadata inside the Partition that contains the Primer Pack. Index Table parsers shall not use the Primer Pack to decode Local Tags within the Index Tables. The Primer Pack shall not cover Local Sets within Essence Containers.

MXF encoders should not accumulate Primer Packs entries across multiple Partitions containing Header Metadata.

The Key of the Primer Pack is given in Table 13.

**Table 13 – Primer Pack Key Value**

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

The value of the Primer Pack is defined in Table 14.

**Table 14 – Primer Pack**

	Item Name	Type	Len	Item UL	Meaning	Default
☐	Primer Pack	Pack Key	16	Table 13	Identifies a Primer Pack	
↔	Length	BER Length	var (see 9.3)		Overall Length of Primer Pack	
	LocalTagEntry Batch	Batch of Local Tag Entry	8+ 18n	06.0E.2B.34 01.01.01.05 06.01.01.07 15.00.00.00	A Batch of Local Tag to UL mappings (see Table 15)	

The LocalTagEntry Batch is a Batch of Local Tag to UL or UUID mappings with the format below. The pairs are in no particular order. The UL or UUID values shall be stored in a Property of type AUID.

There shall be only one entry for each Local Tag in the LocalTagEntry Batch.

For each AUID in the LocalTagEntry Batch, there shall be zero or one dynamically allocated Local Tag (see Section 9.2.2).

**Table 15 – LocalTagEntry Batch**

#	Item Name	Type	Len	Item UL	Meaning	Default
	Number of items	UInt32	4	N/A	The Number of Items in the Batch	N
	Item Length	UInt32	4	N/A	The Length of each Item	18
N	Local Tag	UInt16	2	06.0E.2B.34 01.01.01.05 01.03.06.02 00.00.00.00	The value of the Local Tag. [Note: SMPTE RP 210 definition: A locally unique registry identifier]	
	UID	AUID	16	06.0E.2B.34 01.01.01.05 01.03.06.03 00.00.00.00	The UID of which the Local Tag is an alias	

### 9.2.1 Contents of the Primer

The Primer shall contain all Local Tags used in Header Metadata Sets using 2-byte Local Tag encoding the Partition in which it occurs. The Primer may also contain Local Tags which are not used within the Partition.

### 9.2.2 Local Tag values

Local Tags shall be allocated in the following ranges:

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

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

The dynamically allocated tags are allocated on a Partition by Partition basis. This means that when a file is re-written, the mapping between a dynamic 2-byte Local Tag and an AUID may change.

The algorithm for allocating the Dynamic Tags is not specified. The algorithm shall ensure that all constraints defined for the table written in this specification are met.

The allocation of static Local Tags for MXF Metadata sets is deprecated. All new MXF Set Properties shall use dynamically allocated Local Tag values. Annex H contains the list of all statically assigned MXF Local Tags and their associated ULs.

### 9.2.3 Dark Metadata Support

Metadata values inserted into a file which are unknown to a decoder are called Dark Metadata (see Section 6.3.2).

Metadata extensions to Header Metadata Sets using 2-byte Local Tags shall place their tags in the Primer Pack.

MXF decoders should not attempt to interpret Local Tag values within Sets using other than 2-byte Local Tag encoding and whose Set Key is not understood by the decoder.

Note: Any implementation inserting Header metadata in MXF files which does not conform to the MXF constraints upon SMPTE ST 336 jeopardizes future public specification of that metadata.

MXF decoders are not required to react to Dark Metadata. They should preserve any Dark Metadata in the file to enable later processes to use it.

9.3 Header Metadata Set Coding

An MXF Set shall be an instance of a Class derived from Interchange Object using KLV Set encoding.

Interchange Object shall be the Abstract Superclass of all MXF Sets. It shall be as defined in Annex A.1. It shall contain the required Properties and may contain the optional Properties defined in that section.

Note: Annex A specifies that all MXF Structural Metadata Sets can be extended by adding Optional Properties, that the addition of Required, Encoder Required, Decoder Required or Best Effort Properties must result in the assignment of a new Group UL and thus the definition of a new Structural Metadata Set.

The Header Metadata shall be a sequence of KLV coded MXF Sets which describe the contents of the File Body.

All Structural and Application-Specific Metadata shall be encoded as MXF Local Sets. All Descriptive Metadata should be encoded as MXF Local Sets.

MXF encoders should use 4-byte BER long-form encoding for the Length of all Header Metadata Sets.

MXF encoders shall use 2-byte local length encoding for all MXF Local Sets in which the values of all Properties have a length equal or smaller than 65535 bytes.

MXF encoders shall use BER local length encoding for all MXF Local Sets in which the value of one or more Properties have a length of greater than 65535 bytes.

Note 1: According to SMPTE ST 336, the xxh entry in Table 16 has the value of 13h for BER long or short form encoded length and 53h for 2-byte length.

Note 2: MXF decoders need to comply with Section 6.3.4 for Header Metadata decoding.

Note 3: According to Section 6.3.4, decoders must support BER long-form encoded Lengths occupying up to 9 bytes.

An example of KLV packets is shown in Figure 8.

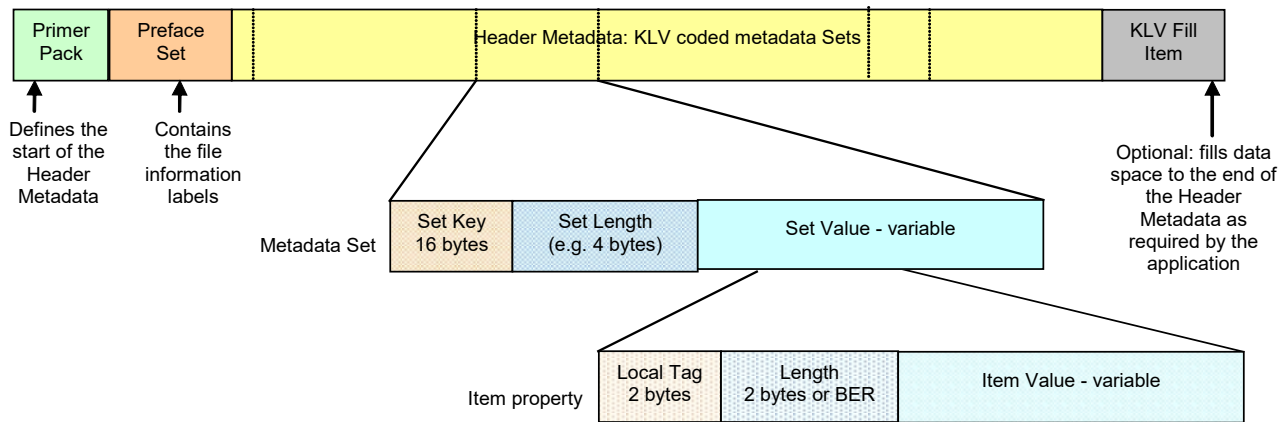


Figure 8 – KLV Data Coding in the Header Metadata

### 9.3.1 Data Model (Informative)

The Structural Metadata is defined in Section 9.4. It defines the Structural Metadata Sets and Properties that need to be included for a complete definition of the File Body.

The definitions of Operational Patterns are given in Section 8. They define the Package and Track complexity as well as other constraints that apply for a complete definition of the File Body.

The Descriptive Metadata is used to define various editorial aspects of the file, for example production and clip information. The basic rules for Descriptive Metadata and the mechanism for how it is added to Header Metadata are defined in Section 9.8.

Application-Specific Metadata is used to exchange application-specific information through MXF files. The basic rules for Application-Specific Metadata and the mechanism for how it is added to Header Metadata are defined in Section 9.7.

### 9.3.2 Strong and Weak References

References are merely a way of allowing one item to refer to another. This may be a one to one relationship implying ownership (Strong Reference) or a many to one relationship (Weak Reference).

Strong References shall be made to the Instance UID of another Set.

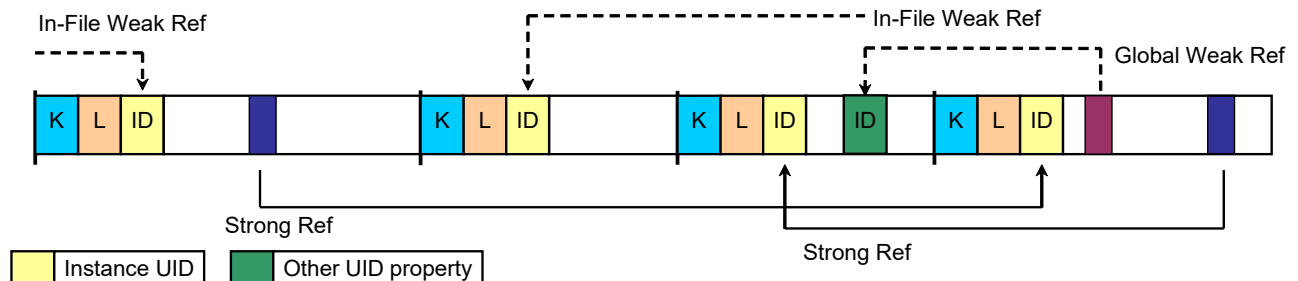
Note 1: This is known as the referencing method. The alternate method of embedding each strongly referred data set into the referring data set (as allowed by the recursive grouping mechanism in SMPTE ST 336) is not used in MXF defined Header Metadata. Its use is discouraged, but not prohibited in Dark Metadata.

Global Weak References shall be made either to another globally unique ID of another Set, to a UL or to a UMID. Global Weak References shall not be made to the Instance UID of another Set.

In-File Weak References shall be made to the Instance UID Property of an MXF Set. Applications that manipulate MXF files shall preserve the values of all Instance UID Properties.

Note 2: SMPTE 377M-2004 did not require preservation of Instance UID values.

Figure 9 illustrates a sequence of KLV packets, each with a unique identification (UID) and the connections between packets.



**Figure 9 – Using UIDs to Connect Metadata Sets in a Data Stream**

The logical model that results from the use of references and other syntactic constructs is defined in Section 9.5 of this specification.

Logical models for Descriptive Metadata Schemes shall follow the same principles (i.e. Strong and Weak References) to represent hierarchical document structures.

Logical models for Application Metadata Schemes should follow the same principles (i.e. Strong and Weak References) to represent hierarchical document structures.

### 9.3.3 Uniqueness of Instance UID values

The Instance UID values used in the MXF Header Metadata are globally unique values.

The following application rules shall apply to their management:

1. Simple repetition of the Header Metadata shall not require the creation of new Instance UID values.
2. Copying of an MXF file shall not require the creation of new Instance UID values.
3. Updates to any existing Header Metadata Set shall not require the creation of a new Instance UID value.

For the rules governing the creation of Generation UID values see Section 7.5.2.

## 9.4 Structural Metadata Semantics

The Header Metadata is generally composed of a sequence of Metadata Sets which are positioned adjacent to each other and may be multiplexed with Fill items. These Sets are connected to each other using Strong or Weak References (Section 9.3.2) in order to provide a hierarchical logical data model. Figure 10 and Figure 12 broadly illustrate the semantics of the Structural Metadata and how it relates to the content of the File Body.

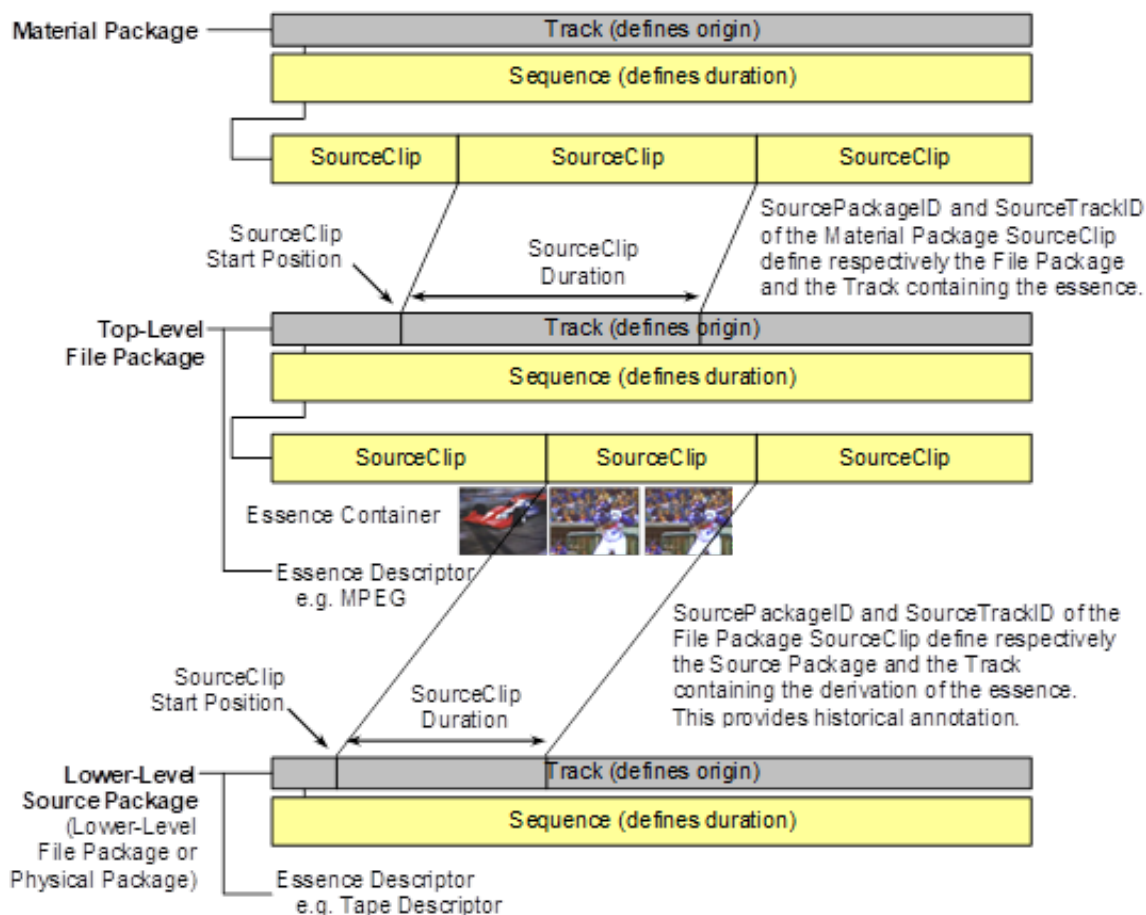
### 9.4.1 Explanation of Figures illustrating the Structural Metadata Semantics (Informative)

Figure 10 shows the main synchronization and referencing features of an MXF file:

1. The Material Package (at the top) is a metadata structure which represents an “output” timeline of the file. If the MXF file is played then this timeline represents the default output that will be seen and heard.
2. The Material Package owns a Track which defines the start and Edit Rate of the output.
3. The Track owns a Sequence which defines the duration of the output.
4. The Sequence, in turn can be divided into one or more Source Clips.
5. The Material Package Source Clip is linked to a Track within a Top-Level File Package. In Generalized Operational Patterns, the Material Package references a Top-Level File Package (see Section 8.3).
6. The Top-Level File Package is a Source Package that contains a File Descriptor which identifies the Essence Container. The Top-Level File Package can also contain a link to another Source Package, which is called a Lower-Level Source Package.
7. A Lower-Level Source Package contains information about how a file was created and possibly contains a Physical Descriptor which could describe the video tape or other physical medium from where the material came. This is historical annotation and can give rise to Lower-Level Source Packages referencing other Source Packages to any depth as long as there is no recursion. See Annex E.5 for more details.



Note: An example of the use of this mechanism could be in the creation of a promotional program by a broadcaster. The Material Package defines the output. The Top-Level File Packages contain material from several television programs which are being promoted. The first Lower-Level Source Packages in turn describe the MXF files used to make the television programs. The next Lower-Level of Source Package could define the tapes used to capture the original material.



**Figure 10 – Header Metadata Packages**

Figure 10 also illustrates the relationship between the Track and the actual Essence in its associated Essence Container. In the centre of the figure, it shows the Essence data which would be KLV encapsulated in MXF using one or more of the Essence Container specifications. In the upper part of the figure is a representation of the Structural Metadata which generally describes the output timeline of the file through the Material Package. The Material Package governs the synchronization and play order of the Source Clips defined in the Top-Level File Package. This is achieved using a number of parallel Tracks each comprising a sequence of one or more Source Clips. Each Source Clip in a Material Package Track can describe the location of the Essence item in the Top-Level File Package. The Top-Level File Package in turn identifies the actual Essence Container through the **EssenceContainerData** Set using the **BodySID** value to link to the Partition(s) containing data of that Essence Container [see Figure 17].

Each Source Clip in a Material Package has a **SourcePackageID** value and a **SourceTrackID** value that identify respectively the Top-Level File Package and the Track which is to be accessed. The Source Clip also has a **Start Position** value and a **Duration** value that accesses the portion of the Track required. The **Start Position** value is the number of Material Package Track Edit Units along the Top-Level File Package Track with the zero Edit Unit number on the Track Set defined by the Zero Point of Top-Level File Package Track Origin value. Clearly, the

Source Clip being extracted from the Top-Level File Package must fall within the total duration of the referenced Track. This reference chain ends when a zero value SourcePackageID is encountered. The same mechanism is used within a Source Package to access Tracks in other Lower-Level Source Packages. However, whereas a Top-Level File Package represents Essence data that may be internal or external to the file, Lower-Level Source Packages are used to annotate the derivation of that Essence.

Annex E defines the two categories of Source Package available in MXF: File Packages and Physical Packages.

Metadata Tracks such as those used for timecode and Descriptive Metadata refer to the Package which contains them. Thus, the Timecode Track in a Material Package defines the timecode for playout of the Material Package (i.e. file playout) and will therefore be continuous. A Timecode Track in a File Package represents the timecode in the Essence Container associated with that File Package. Since the timecode in an Essence Container can be discontinuous, the File Package can require one or more timecode Source Clips to represent the desired timecode values for the Essence Container.

9.4.2 The MXF timing Model

Synchronization between Tracks within any Package is based on the MXF timing model as described below.

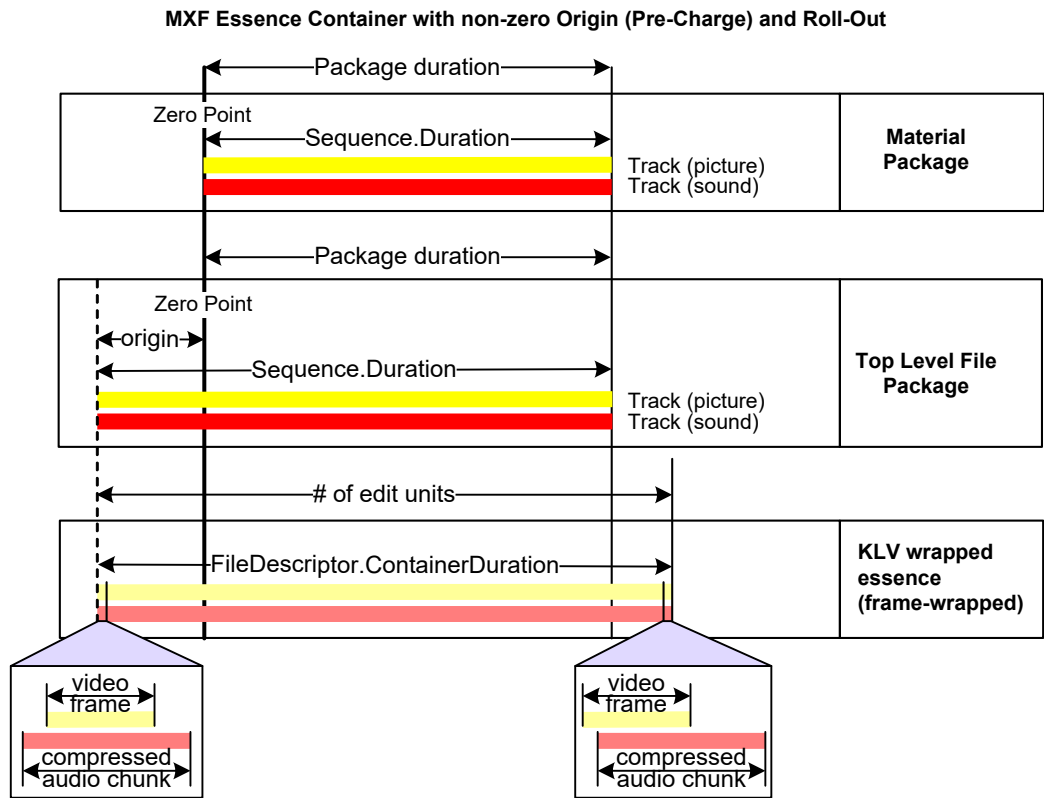


Figure 11 – MXF Timing Model example where all Tracks have the same Edit Rate

Figure 11 shows the relationship between a timeline position and the Zero Point. The Position Property is used to measure elapsed time within the timeline of the Package. In the text below, the term Position refers to some timeline position in the Material Package or the Top-Level File Package. The value Position=0 shall be located at the Zero Point of the Packages.

In most files, Essence Container Tracks will start at Position=0, with an Origin of zero (0). Within a Top-Level File Package:

1. The value of Edit Rate shall be identical for every timeline Essence Track of the Top-Level File Package. The value of Edit Rate of the timeline Essence Tracks of one Top-Level File Package need not match the Edit Rate of the Essence Tracks of the other Top-Level File Packages.
2. The value of Origin shall be identical for every timeline Essence Track of the Top-Level File Package.

The Origin Property of a File Package Essence Track shall be set to a non-negative value equal to the maximum number of Edit Units of stored Essence before the Zero Point.

Note: SMPTE ST 379 requires empty KLV elements to be prepended to the Essence so that there are the same number of Edit Units before the Zero Point on every Track. This means that, according to SMPTE ST 379, empty Essence Elements must be added to the Content Packages in order to make the value of Origins match across all Essence Tracks.

If empty Essence Elements are stored in the Essence Container, the Top-Level File Package describing the Essence Container may signal this with a Filler Component at the start of each of the corresponding Essence Track.

Every stored Essence sample can be identified by the Time Offset from the start of stored Essence. This Time Offset from start of Essence on Track  $n$  is called  $index\_position_n$  and is given by

$$index\_position_n = \frac{Position_n + Origin_n}{EditRate_n}$$

Samples in the Essence Container which have the same value of Position on any Track are synchronized and shall be played at the same time in an MXF decoder.

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

When a MXF decoder plays a Material Package of an MXF file, it synchronizes Essence at  $Position_m$  on each Material Package Track. Given the Top-Level File Package  $f$  and Material Package  $m$ , shall be calculated from the parameters of  $f$  and  $m$  as follows:

$$index\_position_f = \frac{\frac{Position_m}{EditRate_m} \times EditRate_f + Origin_f}{EditRate_f}$$

In files where the Operational Pattern is higher than OP1a, there may be several Source Clips on the Material Package Track. The player application needs to work out the playback position within the Source Clip using the Edit Units of the Material Package Track.

In our example of Material Package  $m$ , let's assume current play position is  $Delta_m$  Edit Units into a Source Clip $_m$ . Source Clip $_m$  has a Property  $StartPosition_m$  that indicates the start of the Source Clip as an offset into the referenced Track of the Top Level Source Package  $f$ . In that case, index position is computed as:

$$index\_position_f = \frac{\frac{StartPosition_m + Delta_m}{EditRate_m} \times EditRate_f + Origin_f}{EditRate_f}$$

This index position may now be used as a lookup into the Index Table to find the byte offset of the Essence Element in the Essence Container.

The Package Duration is defined for Packages containing Essence Tracks

$$Package\_Duration = earliest\_end\_position - latest\_start\_position$$

Where

$$earliest\_end\_position = \max\left(0, \min[end\_position]_{first\_essence\_track}^{last\_essence\_track}\right)$$

and

$$latest\_start\_position = \max\left(0, \max[start\_position]_{first\_essence\_track}^{last\_essence\_track}\right)$$

For Packages not containing Essence Tracks, *earliest\_end\_position* and *latest\_start\_position* are calculated over all Tracks instead of all Essence Tracks.

The normalized Position on all Tracks must be aligned even when they have different values of Edit Rate and Origin.

Timecode is represented by a Track and can be used to annotate the value of Position at any point in the Package timeline. When using timecode to synchronize different streams, it is important to realize that in MXF, timecode is a metadata annotation and a timecode value gets converted to an Edit Unit count value (and hence a Position value), which is the underlying synchronization mechanism.

For a Timecode Track with Origin equal 0, the timecode value at the Zero Point of the Track equals the start timecode value of the first Timecode Component. For a Timecode Track with a single Timecode Component and with origin N, where N greater than 0, the timecode value at the Zero Point of the Track equals the start timecode of the Timecode Component incremented by N units.

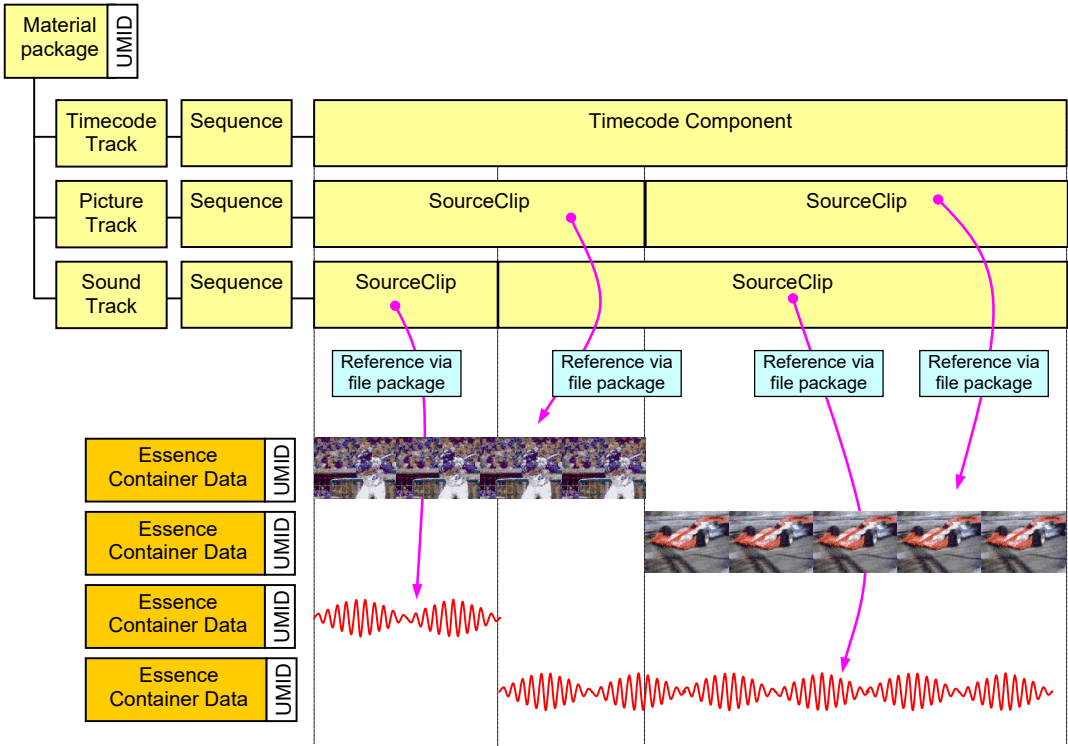


Figure 12 – Logical Header Metadata Structure and its Relation to Essence Data

9.4.3 Relationship between File Packages and Essence Containers

Each File Package shall be related to one Essence Container through an EssenceContainerData Set as illustrated in Figure 12 and in more detail in Figure 17.

There shall be one EssenceContainerData Set for each Top-Level File Package that describes internal Essence or that describes Essence with an associated Index Table that is internal to the file. The EssenceContainerData Set shall be linked to its associated Top-Level File Package through a common Package ID value.

Each EssenceContainerData Set shall have a BodySID value that identifies any Partition(s) in this file that contain the Essence Container.

Each EssenceContainerData Set may also have an IndexSID value that identifies the Partition(s) that contain the Index Table used to index the contents of that Essence Container.

9.5 Structural Metadata Definition

This section specifies the MXF Class structure so that it can be used to define all Operational Patterns. The Normative Annexes of this document define the Structural Metadata sets and their Property definitions.

Note: The Class structure of the MXF Structural Metadata is based on a simplified instance of the AAF metadata Class model.

9.5.1 Header Metadata start

The first Header Metadata Set after the Primer Pack shall be the Preface Set. The Preface Set constitutes the root of the Strong Reference tree of the Header Metadata. The Preface shall directly or indirectly strongly reference all other Header Metadata sets in that Partition.

A Preface Set provides the values of the Operational Pattern UL, a Batch of zero or more Descriptive Metadata Scheme ULs and a Batch containing zero or more Essence Container ULs. The values of these Properties provide an early indication of the complexity of the file structure and its Essence Containers.

9.5.2 Generic Class diagram (Informative)

The Structural Metadata Class model is represented below in diagrammatic form using UML notation.

Each Class is instantiated as KLV coded Metadata Set.

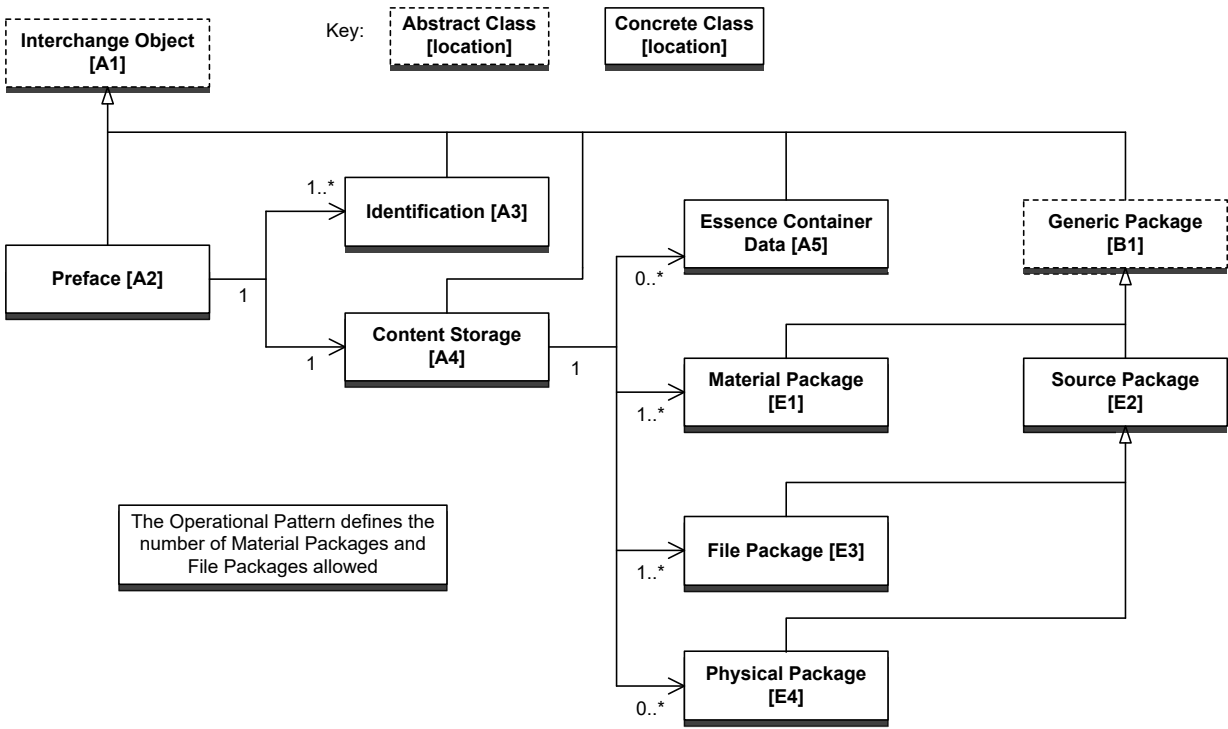


Figure 13 – Root Metadata Sets for Structural Header Metadata

Figure 13 shows the root Header Metadata Set structure for an MXF file.

An MXF parser would find a Partition Pack within the file and then look to see if there was Header Metadata in that Partition. It would then look for the Preface Set which defines the version and contents of the Structural Metadata. The Preface Set references one or more Identification Sets which contain human readable information about the tool(s) used to create or modify the MXF file. A new instance of this Identification Set is created every time the contents of the Header Metadata is modified and stored.

The Preface Set also references the Content Storage Set which identifies all the Packages (Material, File or Physical) and the EssenceContainerData Sets in the Structural Metadata. The relationship between these Packages is given in Section 9.4 and shown diagrammatically in Figure 10. In order to define these Packages (Material, File and Physical), a Generic Package Class has been defined from which two concrete Subclasses have been derived: Material Packages and Source Packages (e.g. File and Physical Packages).

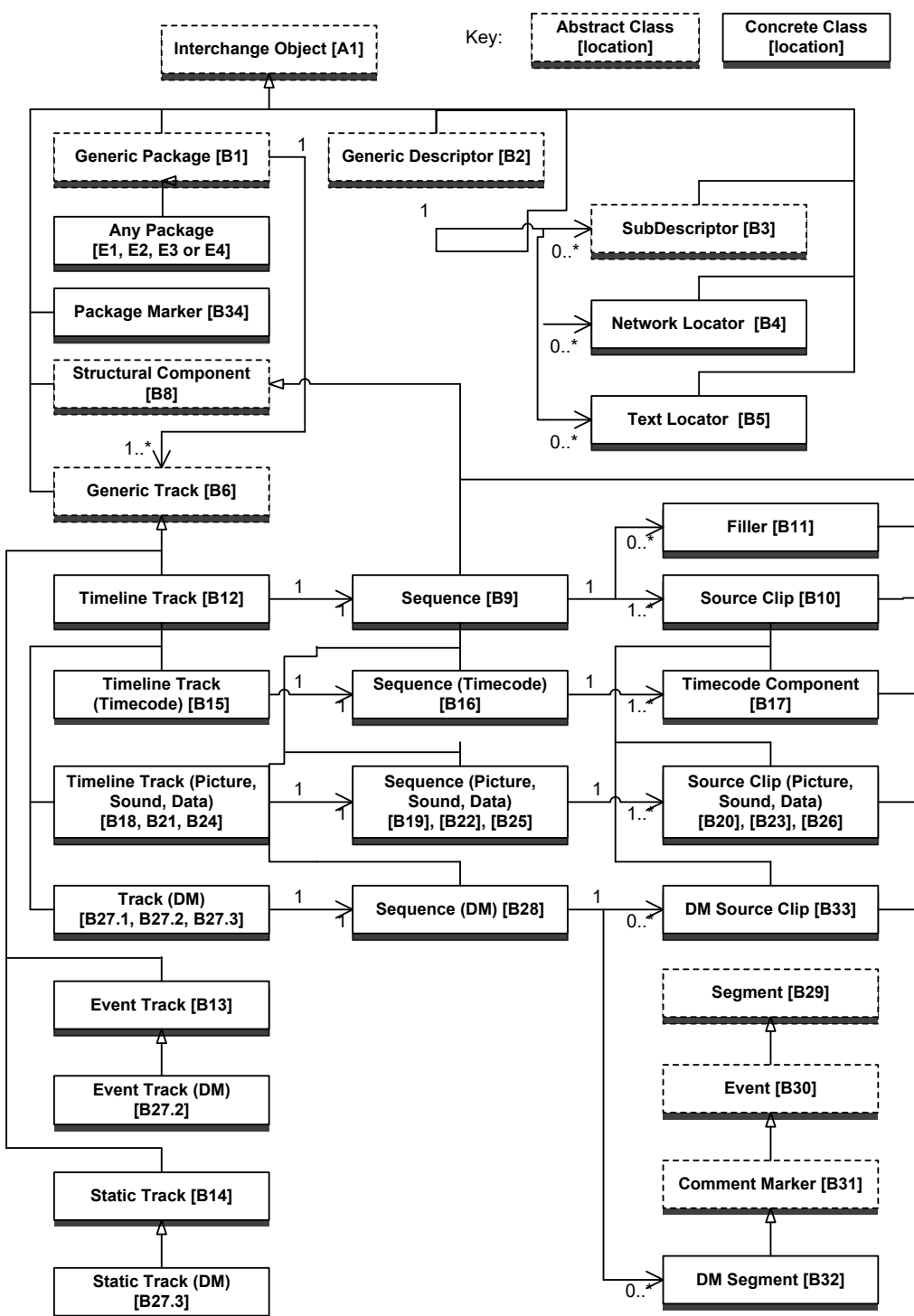


Figure 14 – Diagram of a Generic Package in the Structural Header Metadata

### 9.5.3 Material Package

The Material Package shall be a Subclass of the Generic Package with the following semantics:

1. The number of Material Packages in a file shall be defined by an Operational Pattern specification.
2. The Tracks of a Material Package shall define the “output timeline” and therefore the playout edit rate of the file (Figure 10). The Edit Rates defined in the Material Package shall be used even if a track in a subordinate Source Package specifies a different edit rate.
3. The number of Picture Tracks, Sound Tracks and Data Tracks shall be controlled by the Operational Pattern specification and by the Top-Level Source Package(s) which are associated with the Material Package.
4. The Source Clips of any Essence Track in a Material Package shall link to an Essence Track in a Top-Level Source Package.
5. There shall be zero or one Timecode Track in a Material Package. Operational Pattern specifications may require a Timecode Track to be present.
6. There shall be only one Timecode Component in a Material Package Timecode Track (i.e. continuous output timecode).
7. The value of the Origin Property of all Essence Tracks and all Timecode Tracks in a Material Package shall be zero.
8. There shall be zero Descriptor Sets in the Material Package.
9. There shall be zero EssenceContainerData Sets linked to the Material Package.
10. Any Descriptive Metadata referenced by or contained in the Material Package shall relate to the “output timeline” defined by this Package.

### 9.5.4 Source Package

The Source Package shall be a Subclass of the Generic Package with the following semantics:

1. The Source Package shall have a strong reference to a Descriptor to describe Essence.
2. Depending on the type of their Descriptors, Source Packages are known as File Packages or Physical Packages. Further details are given in Annex E.4.

### 9.5.5 Top-Level File Packages

Top-Level File Packages shall be Source Packages that have the following additional semantics:

1. The number of Top-Level File Packages in an MXF file and the Descriptors they contain (by direct or indirect Strong reference) shall be controlled by the Operational Pattern specification.
2. The Tracks of these Top-Level File Packages shall represent the “input timeline” of the file (Figure 10).
3. There shall be one or more Picture Tracks for each Picture Element of the Essence Container. There shall be one or more Sound Tracks for each Sound Element of the Essence Container. There shall be one or more Data Tracks for each Data Element of the Essence Container. There shall be one or more Picture, Sound or Data Track for each Compound Element of the Essence Container. Essence Container specifications may define additional provisions about the number of Tracks for specific Essence Containers.



Note 1: The term Compound Element is defined in SMPTE ST 379.

Note 2: SMPTE ST 377-1 does not define the linkage of Tracks to individual components within Elements that are of the same Essence type (i.e. Data, Sound or Picture). Readers are advised to check the respective Essence Container specification or for other SMPTE Engineering Documents that define the mechanism.

4. The number of Picture Tracks, Sound Tracks and Data Tracks shall also be controlled by the Operational Pattern specification and by the Essence Container or external Essence file which is associated with the Top-Level File Package.
5. The Source Clips of any Track in Top-Level File Package may associate with a Track in another Lower-Level Source Package (i.e. a Source Package not referenced by a Material Package).
6. There shall be zero or more Timecode Tracks in a Top-Level File Package. Operational Pattern Specifications may require a Timecode Track to be present.
7. There shall be one or more Timecode Components in a Top-Level File Package Timecode Track.
8. There shall be one File Descriptor Set in a File Package. This may be a Multiple Descriptor strongly referencing two or more File Descriptors.
9. Each Essence Element in the Essence Container or external file shall be described by one or more File Descriptors. If the Essence Element is described by more than one File Descriptor, the Package shall have a Multiple Descriptor that references all individual File Descriptors for this Essence Element and all File Descriptors of all other Essence Elements in the Essence Container or external file. If the Essence type is not known by the MXF encoder, MXF encoders should use Data Elements and Data Descriptors to encapsulate the Essence.
10. For each Essence Track there shall be one or more File Descriptors.

Note: Some Essence Container specifications define File Descriptors additional to those specified in Annex F.

11. There shall be one Top-Level File Package for each external Essence file.
12. All Essence Tracks of a Top-Level File Package shall have the same value of Edit Rate. All other Tracks of a Top-Level File Package should have the same value of Edit Rate as the Essence Tracks.
13. All Essence Tracks of a Top-Level File Package shall have the same value of Origin. The value of Origin shall equal the count of Edit Units before the first displayed Edit Unit of the Essence Container. If the material is intercoded, the Edit Units before the first displayed Edit Unit of the Essence Container shall include any necessary Pre-Charge.

Note: Different Top-Level File Packages can have different values of Origin.

14. Material before the Zero Point of the Top-Level File Package shall not be played.
15. The value of the Duration Property of the Sequence Set of Top-Level File Package Essence Tracks shall equal the count of Edit Units starting at the first Edit Unit of the Essence Container to the last played Edit Unit in the Essence Container. It shall be the same for all Top-Level File Package Essence Tracks.
16. The value of the Container Duration Property of a File Descriptor shall equal the count of Edit Units starting from the first Edit Unit in the first non-empty Essence Element to the last Edit Unit in the last non-empty Essence Element that is described by the Essence Track to which the File Descriptor is associated.
17. The value of the Container Duration Property of the Multiple Descriptor shall equal the count of Edit Units starting at the first Content Package with a non-empty Essence Element (across all Essence Elements in the Container) to the last Content Package with a non-empty Essence Element (across all Essence Elements in the Container).
18. Any Descriptive Metadata referenced by or contained in the Top-Level File Package shall relate to the content of the Essence Container or external Essence file associated with this Source Package.

### 9.5.6 Lower-Level Source Packages

Lower-Level Source Packages shall be Source Packages that have the following semantics:

1. The Tracks of a Lower-Level Source Package document the derivation or history of Source Clips of the source Essence (Figure 10).
2. The number of Lower-Level Source Packages in a file is controlled by the application writing the file. It shall not be subject to any Operational Pattern specification.
3. There shall be zero or more Timecode Tracks in a Lower-Level Source Package.
4. There shall be one or more Timecode Components in a Lower-Level Source Package Timecode Track.
5. The number of Picture Tracks, Sound Tracks and Data Tracks shall not be subject to any Operational Pattern specification as they constitute historical annotation only.
6. All Tracks of a Lower-Level Source Package should have the same value of Edit Rate.
7. Any Descriptive Metadata referenced by or contained in a Lower-Level Source Package shall relate to the content of the original source Essence that this Package describes.
8. If a Lower-Level Source Package is a File Package, it shall have a File Descriptor. This File Descriptor may be a Multiple Descriptor strongly referencing two or more File Descriptors.
9. If a Lower-Level Source Package is a Physical Package, it shall have a Physical Descriptor.

### 9.5.7 Relationship between the Packages and SourcePackageID / SourceTrackID

The Content Storage Set contains a Batch of strong references to every Package in the file. At least one of these shall be a Material Package. This Package shall contain a number of Tracks, each of which has a Sequence, which in turn will have a number of Source Clips.

File Package Essence Tracks shall have a Track Number whose value is defined by the Essence Container specification.

Note: According to Annex B.7, each Track must have a unique Track ID to which a Source Clip can refer.

The Source Clips shall contain:

1. A SourcePackageID which identifies the Package which this Source Clip references
2. A SourceTrackID which identifies the Track within the Package which this Source Clip references
3. A Start Position which identifies the Edit Unit in the Package Track (measured in Edit Units defined according to the Edit Rate of the Track which owns the Source Clip relative to the Zero Point of the Package)
4. A Duration which identifies the length of the Essence Element in the Package Track (measured in Edit Units defined according to the Edit Rate of the Track which owns the Source Clip)

The same mechanism shall be used to relate each Source Clip in a Material Package to the timeline of a Source Package Track, and each Source Clip in a Source Package to the timeline of a Lower-Level Source Package Track. The chain of Source Package references shall be terminated by setting the SourcePackageID and SourceTrackID to zero.

The relationships are shown diagrammatically for the Material Package and File Package in Figure 15 and Figure 16.

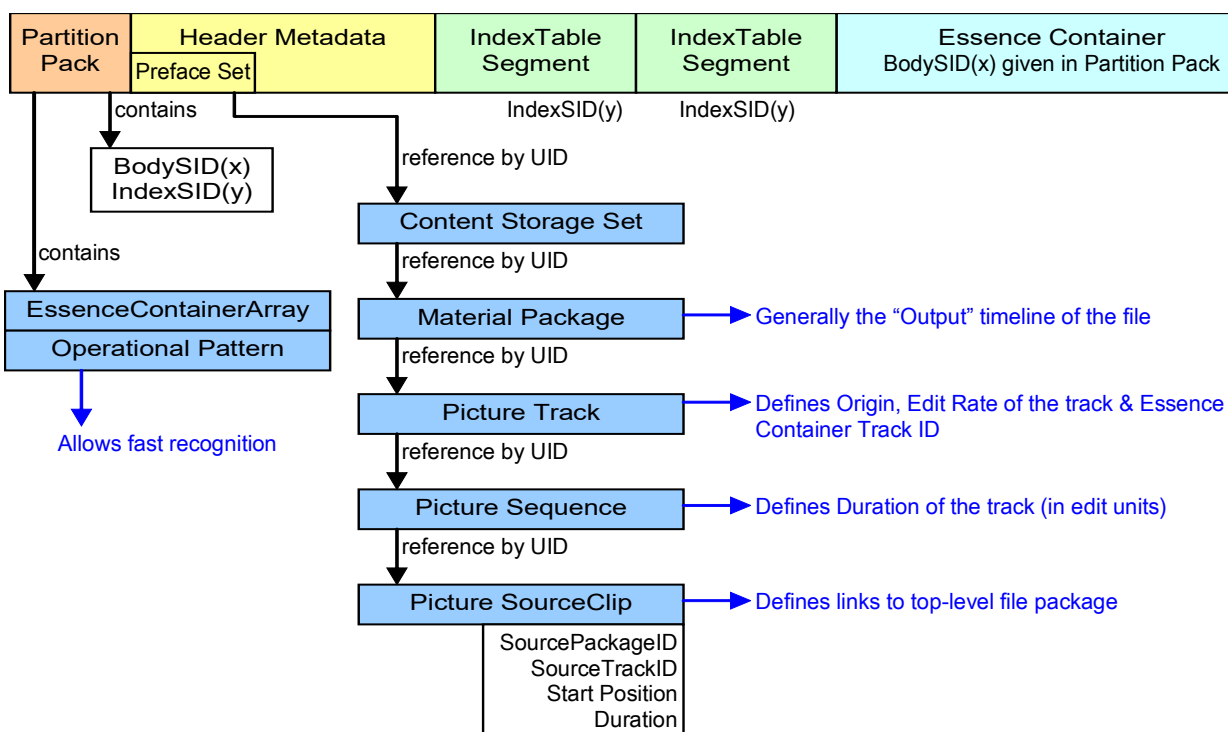


Figure 15 – Relationship between Material Package references

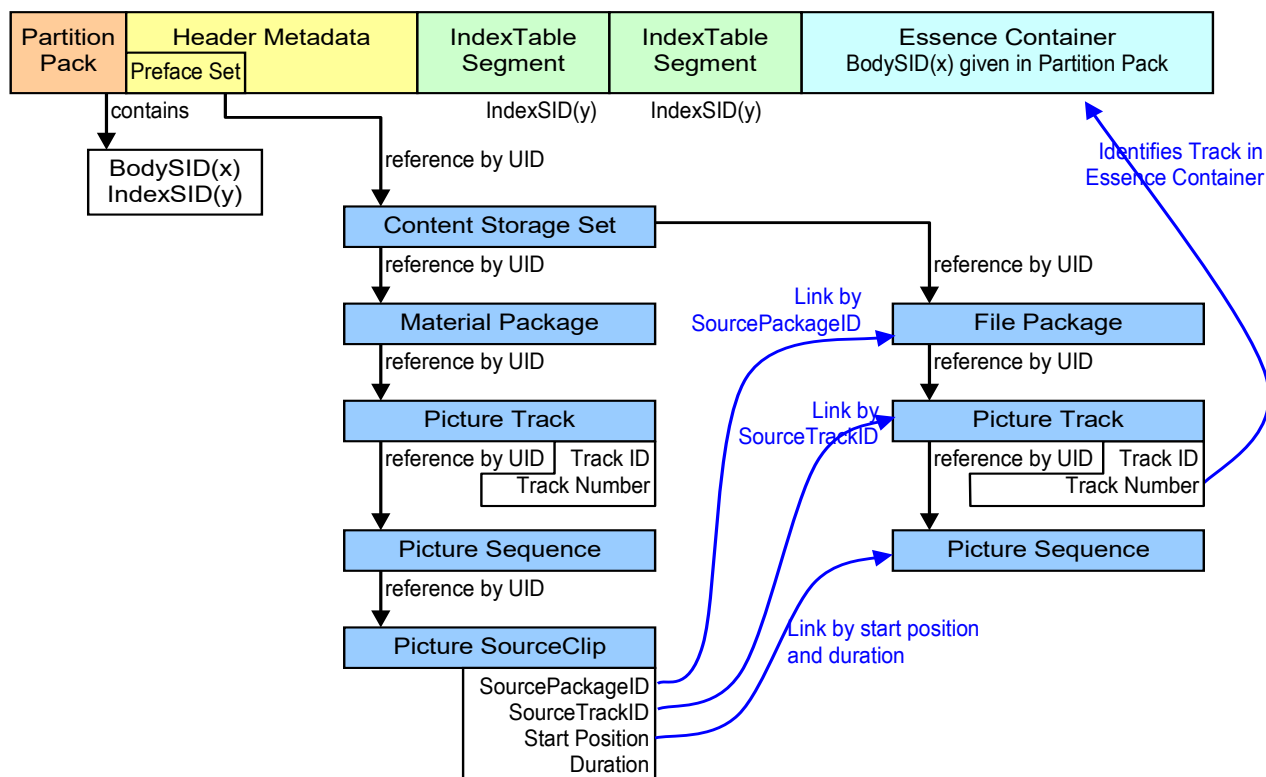


Figure 16 – Relationship between Material Package SourcePackageID / SourceTrackID and File Package

9.5.8 Relationship between the BodySID and IndexSID

The Content Storage Set contains a Batch of strong references to Essence Container Data Sets, each of which defines the relationship between BodySID and IndexSID for one Top-Level File Package.

The UMID of the File Package is used to link it to one of the EssenceContainerData sets which in turn defines the BodySID and IndexSID values which must be used for that Package. This is shown diagrammatically in Figure 17.

If both the IndexSID and BodySID of an EssenceContainerData Set equal zero (0), MXF decoders shall react identically to the case in which the EssenceContainerData Set is absent from the file.

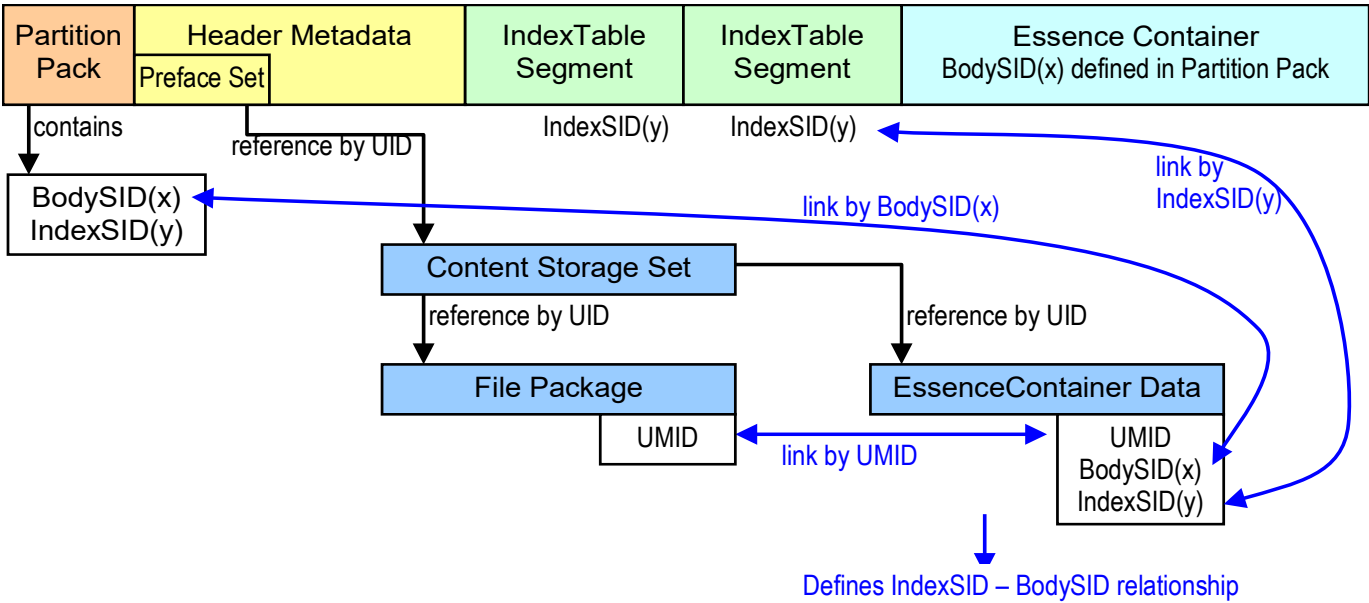


Figure 17 – Relationship between BodySID and IndexSID

9.5.9 Scope of the Track ID values

The scope of all Track ID values is limited to the Package in which the Track occurs. Therefore a Material Package and a Source Package may use the same Track ID value for the Picture Track, but each Track ID is valid only in the appropriate Package.

9.6 Structural Header Metadata Implementation

All Structural Header Metadata Objects shall be implemented as MXF Local Sets. All Object Properties in KLV coded Local Sets shall be encoded according to Section 6.3.5.

9.6.1 KLV Key values for Structural Metadata Sets

The common Key structure for all the Structural Header Metadata Objects coded as Local Sets in this document shall be defined as follows:

**Table 16 – Common Key Value for the Structural Metadata Sets**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets and packs
6	Registry Designator:	xxh	Local Sets: 2-byte Local Tags with either 2-byte length (default) or BER encoded length
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	vvh	Registry Version in which the Key of the specific Set first appeared
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	01h	MXF / AAF Association Structural Metadata Sets
12	Structure version	01h	Structure Version 1
13	Structure Kind	01h	MXF / AAF Association compatible sets and packs
14	Set Kind (1)	yyh	MXF Set Definition (see Table 17)
15	Set Kind (2)	zzh	MXF Set Definition (see Table 17)
16	Reserved	00h	Reserved

The definition of bytes 14 and 15 of the keys for the encoding of all Structural Header Metadata Sets defined in this specification is given in Table 17.

Note 1: See Section 9.3 for further KLV encoding details. In particular, MXF encoders must use 2-byte local length encoding for all sets in which the values of all Properties have a length equal or smaller than 65535 bytes and BER local length encoding otherwise.

**Table 17 – Key Values for Structural Metadata Sets**

Set Name	Byte 14	Byte 15	Detailed Set Definition (Annex Reference)	Encoding Constraints (Byte 6 of the KLV Key)
Preface	01h	2Fh	A.2	53h only
Identification	01h	30h	A.3	53h only
Content Storage	01h	18h	A.4	53h only
Essence Container Data	01h	23h	A.5	53h only
Material Package	01h	36h	E.1	53h only
Source Package (File, Physical)	01h	37h	E.2 (E.3, E.4)	53h only
Timeline Track (all cases)	01h	3Bh	B.12 ( B.15, B.18, B.21, B.24 and B.27.1)	53h only
Event Track (DM)	01h	39h	B.13 ( B.27.2)	53h only
Static Track (DM)	01h	3Ah	B.14 ( B.27.3)	53h only
Sequence (all cases)	01h	0Fh	B.9 ( B.16, B.19, B.22, B.25, B.28)	53h only
Source Clip (Picture, Sound, Data)	01h	11h	B.10 ( B.20, B.23, B.25)	53h only
Timecode Component	01h	14h	B.17	53h only
DM Segment	01h	41h	B.32	53h only
DM Source Clip	01h	45h	B.33	53h only
Filler	01h	09h	B.11	53h or 13h
Package Marker Object	01h	60h	B.34	53h or 13h
File Descriptor	01h	25h	F.2	53h only
Generic Picture Essence Descriptor	01h	27h	F.4.1	53h only
CDCI Essence Descriptor	01h	28h	F.4.2	53h only
RGBA Essence Descriptor	01h	29h	F.4.3	53h only
Generic Sound Essence Descriptor	01h	42h	F.5	53h only
Generic Data Essence Descriptor	01h	43h	F.6	53h only
Multiple Descriptor	01h	44h	F.3	53h only
Network Locator	01h	32h	B.4	53h only
Text Locator	01h	33h	B.5	53h only
Application Plug-In Object	01h	61h	C.2	53h or 13 h
Application Referenced Object	01h	62h	C.3	53h or 13h

Note 2: In the object model, there are a number of Abstract Superclasses. However, they are never encoded as Metadata Sets and their SMPTE Universal Labels are not included in this Table, but in Table 19.

Note 3: Other SMPTE standards (such as MXF Essence mappings) define additional MXF Structural Metadata Set keys. SMPTE ST 395 registers the Keys of all MXF Header Metadata Sets and identifies the SMPTE Standard that defines their semantics and use.

### 9.6.2 Universal Labels for Abstract Structural Metadata Groups

The common Universal Label structure for all the Abstract Structural Header Metadata Groups in this document shall be defined as follows:

**Table 18 – Common Universal Label Value for all Abstract Structural Metadata Groups**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets & packs
6	Registry Designator:	7fh	Abstract Groups, no KLV encoding syntax specified
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	vvh	Registry Version in which the Key of the specific Group first appeared
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	01h	MXF / AAF Association Structural Metadata Sets
12	Structure version	01h	Structure Version 1
13	Structure Kind	01h	MXF / AAF Association compatible sets & packs
14	Set Kind (1)	yyh	MXF Set Definition (see Table 17)
15	Set Kind (2)	zzh	MXF Set Definition (see Table 17)
16	Reserved	00h	Reserved

The definition of bytes 14 and 15 of the Universal Labels for all Abstract Structural Header Metadata Groups defined in this specification is given in Table 19.

**Table 19 – Universal Label Values for Abstract Structural Metadata Groups**

Set Name	Byte 14	Byte 15	Detailed Abstract Group Definition
Interchange Object	01h	01h	A.1
Generic Package	01h	34h	B.1
Generic Descriptor	01h	24h	B.2
SubDescriptor	01h	59h	B.3
Generic Track	01h	38h	B.6
Structural Component	01h	02h	B.8
Segment	01h	03h	B.29
Event	01h	06h	B.30
Comment Marker	01h	08h	B.31
Application Object	01h	66h	C.1
File Descriptor	01h	25h	F.2

Note 1: Abstract Groups are never encoded as Metadata Sets.

Note 2: Other SMPTE standards can define additional Abstract MXF Metadata Groups. SMPTE ST 395 registers the Universal Labels all Abstract MXF Header Metadata Groups and identifies the SMPTE Standard that defines their semantics and use.

## 9.7 Application Metadata Plug-Ins

### 9.7.1 General (Informative)

This section 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. Each such extension is associated with an Application Metadata Scheme, identified by a SMPTE registered UL.

Other SMPTE Engineering documents can define additional extension methods.

The Application-Specific Metadata plug-in mechanism allows applications within an Application Environment to add metadata to any MXF Set in the Header Metadata (except Application Metadata Plug-in instances) in a mutually independent way. This means any application can add metadata to an MXF file such that:

1. An Application Metadata Scheme specification can define Subclasses of all MXF defined Structural or Descriptive Metadata classes (except the Application Object Class itself).
2. Each Application-Specific Metadata instance identifies the Application Metadata Scheme to which it complies.
3. The Preface signals all Application Metadata Schemes of which there are instances within the Header Metadata.
4. Each Application-Specific Metadata instance identifies the specific Application Environment with which it is associated.
5. The Application-Specific Metadata associated with different Application Environments is mutually orthogonal.
6. The Application-Specific Metadata associated with all or any specific Application Environment can be easily and completely removed from the file, even if the Application-Specific Metadata is Dark to the simple application performing the removal.

The Sets for the Application-Specific Metadata plug-In mechanism are defined in Annex C.

All Properties that directly attach Application Metadata Plug-in instances to MXF Structural Metadata are optional. This means that MXF applications can ignore Application Metadata Plug-In instances.

### 9.7.2 Application Metadata Scheme Specification

For each Application Metadata Scheme, there shall be an Application Plug-in Scheme specification. An MXF Application Plug-in Scheme specification should be a public standard from an internationally accredited standards body. This is to ensure that all Application Plug-in Schemes can be used for interchange.

In addition to the Application Metadata Plug-In payload elements and semantics, Application Metadata Scheme specifications shall define the Application Scheme Label.

For further requirements on Application Metadata Scheme specifications see Annex C.

### 9.7.3 Generic Universal Label for the MXF Application Metadata Schemes

The Generic UL below shall be used to identify MXF Application Metadata Schemes.

There may be non-MXF Application Metadata Schemes which use the MXF Application-Specific Metadata plug-in mechanism. These Application Metadata Schemes may have different Label values to the ones given here. All MXF Application Metadata Schemes shall be identified with the Universal Label defined below.



Note: This means that all Application Metadata Schemes that are identified by a Label conforming to the definition below are MXF Application Metadata Schemes.

**Table 20 – Generic Universal Label for MXF Application Metadata Schemes**

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

#### 9.7.4 Plug-In Mechanism

An MXF file may contain Application-Specific Metadata from zero or more Application Metadata Schemes and associated with zero or more Application Environments.

Each Structural or Descriptive MXF Header Metadata Set may contain (by Strong Reference to Application Plug-In Objects) Application-Specific Metadata that is associated with zero or more Application Environments.

The Application-Specific Metadata instance that extends or applies to a specific Structural or Descriptive Header Metadata Set shall be encoded as an Application Plug-In Object (see Annex C.2) that is strongly referenced from the optional Application Metadata Plug-Ins Property of the Metadata Set.

The Object Class Property of the Interchange Object shall be present in all instances of Application Plug-In Object where the extension constitutes a new Class. It shall not be present in instances of Application Plug-In Object where the extension does not constitute a new Class. When present, value of the Object Class Property shall equal the Class identifier of the new Class (i.e. the KLV Key of the extended Header Metadata Set).

When present in Application Plug-In Object, the value of the Base Class Property Property shall equal the Class identifier value of the immediate Superclass for the extended Set that is defined in an MXF specification.

The Application Plug-In Instance ID Property of the Application Plug-In Object is of type UUID. Its value shall be the unique identifier of the Application Metadata Plug-in instance.

The Application Scheme Property of the Application Plug-In Object is of type UL. Its value shall contain the Label of the Application Metadata Scheme.

The Application Schemes Batch of the Preface Set shall contain all Application Metadata Scheme Labels that are present in the Header Metadata instance. The Application Schemes Batch of the Preface may only be omitted if there are no Application Plug-in instances in the Header Metadata. It shall be present in all other cases.

The Application Environment ID Property of the Application Plug-In Object is of type UTF-16 string. If present, its value shall be the RFC 3986 URI identifier of the Application Environment to which the Application-Specific Metadata applies. All Application Plug-In Objects that are associated to the same Application Environment shall have the same Application Environment ID Property value.

An Application Environment may, for example, identify a specific product, a number of products of a specific vendor, a domain of applications with specific functionality (e.g. broadcast playout) or a specific user application (e.g. the specific production facility of a specific broadcaster).

In addition to the Properties described above, the Application Plug-In Object shall carry all Required, Encoder Required and Best Effort Properties of the extension. It may carry all Optional and Decoder Required Properties of the extension.

The instances of all classes that are connected to the Application Plug-In Object by direct or indirect Strong Reference shall be encoded as Application Referenced Object (see Annex C.3).

The Object Class Property shall be present in all instances of Application Referenced Object. Its value shall equal the Class identifier value of the strongly referenced Object that is contained in the Application Referenced Object instance.

When present in Application Referenced Object, the value of the Base Class property shall equal the Class identifier of immediate Superclass (defined in an MXF specification) for the Set that is contained in the Application Referenced Object instance.

The Application Referenced Object has a required Property called Linked Application Plug-In Instance ID. This Property constitutes a Global Weak Reference to the Application Plug-In Object to which the Application Referenced Object is connected by direct or indirect Strong Reference. The value of the Linked Application Plug-In Instance ID Property shall equal the value of the Application Plug-In Instance ID Property of the Application Plug-In Object to which the Application Referenced Object is connected by direct or indirect Strong Reference.

Note: The Global Weak Reference to the Application Plug-In can be resolved within the Header Metadata instance, except in situations where the MXF file has been manipulated by applications that eliminate some, but not all of the Header Metadata Sets that are dark to them.

In addition to the Properties described above, the Application Referenced Object shall carry all Required, Encoder Required and Best Effort Properties according to its Class definition. It may carry all Optional and Decoder Required Properties of the extension according to its Class definition.

Unless specified otherwise by the Application Metadata Scheme specification, there may only be In-File Weak References between Application Plug-in instances in the case where both

1. The Application Plug-in instances are part of the same Application Metadata Scheme (i.e. share the same value of the Application Scheme Property) and
2. The Application Plug-in instances are associated with the same Application Environment (i.e. share the same value of the Application Environment ID Property).

Unless specified otherwise by the Application Metadata Scheme specification, there shall be no In-File Weak References between Application Plug-in instances in any other case.

If the Application Metadata Scheme specification allows In-File Weak References between Application Plug-in instances such that 1 or 2 are not satisfied, it shall define the behavior of MXF decoders in case those In-File Weak References cannot be resolved.

The inheritance hierarchy of the Structural Metadata Sets for the Application-Specific Metadata Plug-in Mechanism is illustrated in Figure 18 using UML notation. Any Structural MXF Header Metadata Set except the Application Plug-In Object and Application Referenced Object can strongly reference zero or more Application Plug-In Objects that extend the Structural or Descriptive MXF Header Metadata Set. Application Referenced Objects can strongly reference further Application Referenced Objects.

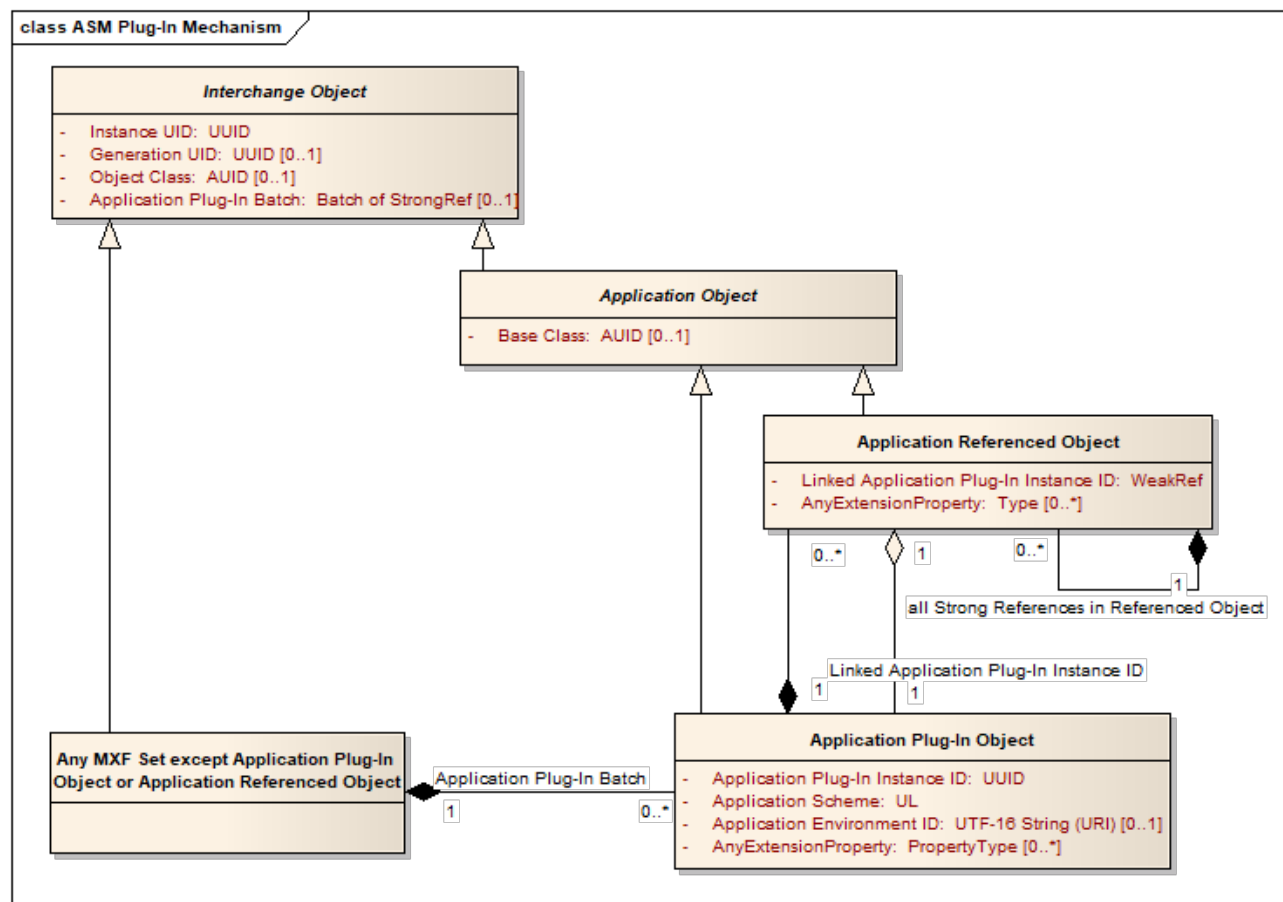


Figure 18 – Application-Specific Metadata Plug-in Mechanism

### 9.7.5 Simple Application Metadata Plug-In Instance Removability

The required Application Environment ID, Application Plug-In Instance ID and Linked Application Plug-In Instance ID Properties uniquely identify all Application Objects that are associated with a specific Application Environment. They can be used to enable simple applications to completely remove specific Application Metadata Plug-ins from the Header Metadata, even if the structure and contents of the Application-Specific Metadata is Dark to the application.

In order to remove all Application Metadata Plug-Ins that are associated with a specific Application Environment and a specific Application Metadata Scheme, a simple application shall remove all Application Metadata Plug-Ins in which the Application Environment ID and the Application Scheme Properties share the same values.

In order to remove all Application Metadata Plug-Ins that are associated with a specific Application Environment, a simple application shall remove all Application Metadata Plug-Ins where the Application Environment ID shares the same value.

#### 9.7.6 Simple Application Metadata Plug-In Instance Removal Implementation (Informative)

In order to remove an Application Metadata Plug-In, a simple application would:

1. remove the value of the Application Scheme Property of the Application Plug-In Object instance from the Application Schemes batch of the Preface, provided there is no other Application Plug-In Object instance with an Application Scheme Property that shares the same value, but is associated with a different Application Environment (signaled by a different value in the Application Environment ID),
2. remove all Application Referenced Object Sets that have a Linked Application Plug-In Instance ID Property value that equals the Application Plug-In Instance ID Property of the Application Plug-In Object and
3. remove the Application Plug-In Object.

#### 9.7.7 Use of the Application Metadata Plug-In Mechanism

All extensions to MXF Structural Metadata for which

1. the behavior of applications that implement the metadata can be uniquely defined such that the definition applies to all use cases (i.e. is use-case independent)

and

2. the metadata is required to satisfy interchange requirements of multiple users or manufacturers

should be defined as MXF Structural or MXF Descriptive Metadata.

All extensions to MXF Structural Metadata for which

- i. the behavior of applications that implement the metadata cannot be uniquely defined

or

- ii. there are multiple independent use cases (or applications) of the information

or

- iii. the Application-Specific Metadata is private or not required to satisfy one or more interchange requirements of multiple users or manufacturers

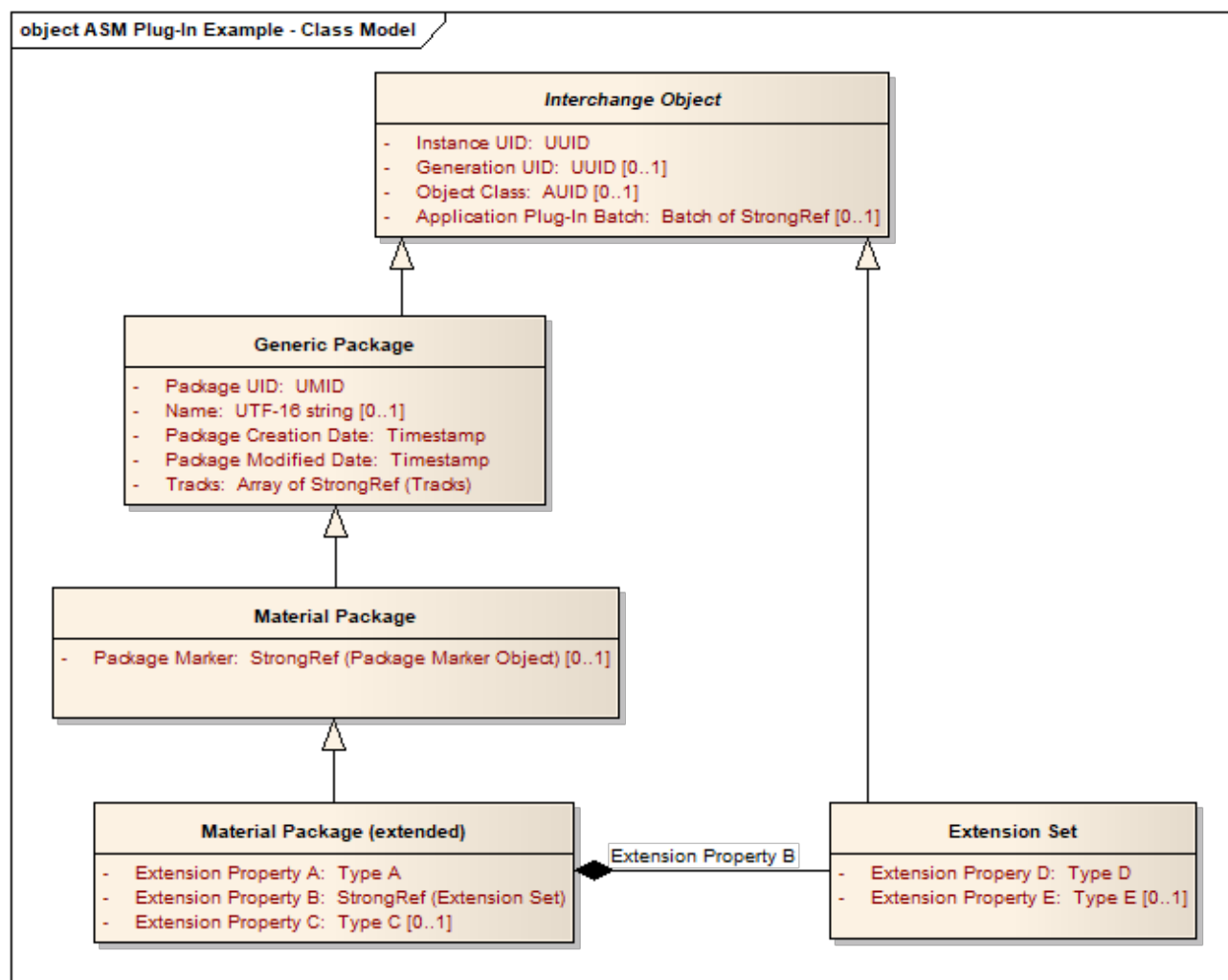
should be specified as Application Metadata plug-in or use alternative extension methods supporting independent extensibility that may be defined in other SMPTE Engineering documents.

The Application-Specific Metadata plug-in mechanism constitutes a universally usable tool to identify specific use cases (or applications) and to allow multiple instances of the same or different Application Metadata Schemes that are targeted at the same or different use cases to coexist within an MXF file.

#### 9.7.8 Application-Specific Metadata Plug-In Mechanism Example (Informative)

This section explains the use of the Application-Specific Metadata Plug-In mechanism via an example.

According to the Class diagram in Figure 19, the extension defines a Subclass of Material Package called Material Package (extended) that adds three additional Properties. Of these three Properties, Extension Property A is a Required Property of data type A, Extension Property B is a Required Property that constitutes a Strong Reference to a Subclass of Interchange Object that is called Extension Set, and Extension Property C is an Optional Property of data type C. Extension Set extends Interchange Object by two Properties. Of these two Properties, Extension Property D is a Required Property of data type D and Extension Property E is an Optional Property of data Type E.



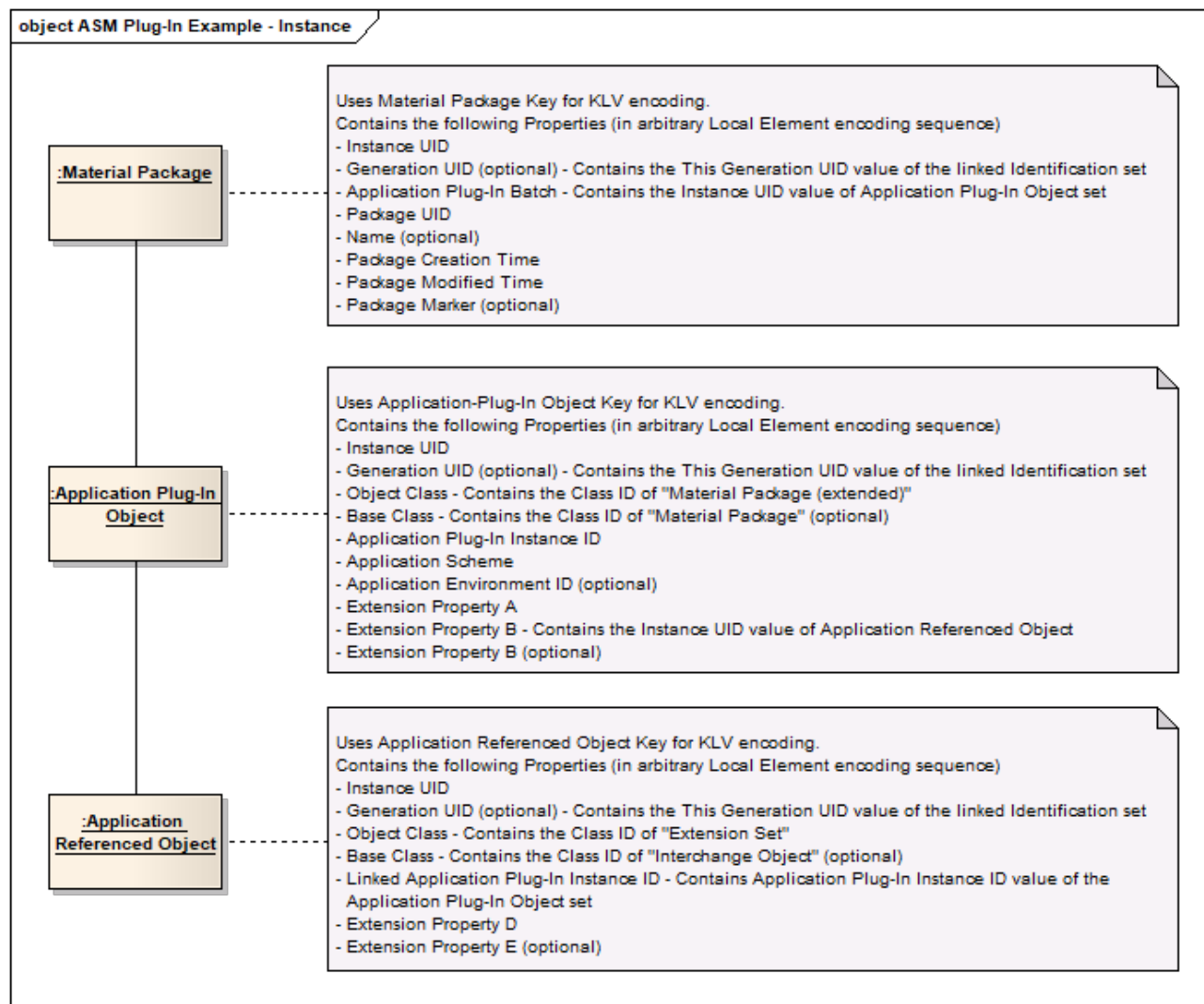
**Figure 19 – Class Diagram for the Application-Specific Metadata Plug-in Mechanism Example**

Figure 20 illustrates the KLV encoding of an instance of the Class model from Figure 19 according to the rules of the Application-Specific Metadata Plug-In mechanism.

The Material Package is encoded using the Material Package Key and holding all properties according to the definition of Annex E.1.

The Class ID of the Material Package (extended) Subclass and all of its additional Properties are encoded in the Application Plug-In Object, along with the additional signaling metadata that is required for the function of Application-Specific Metadata Plug-In mechanism.

The new class Extension Set, which is strongly referenced through Extension Property B, is encoded as Application Referenced Object containing the Class ID of Extension Set and all of its Properties, along with the additional signaling metadata that is required for the function of the Application-Specific Metadata Plug-In mechanism.



**Figure 20 – Object Diagram for the Application-Specific Metadata Plug-in Mechanism Example**

The extension defines a new Subclass of Material Package that has a new Class ID. However, the use of the Application-Specific Metadata Plug-In mechanism assures that MXF decoders that do not implement the extended Subclass or that do not implement the Application-Specific Metadata Plug-In mechanism recognize a known MXF Set that contains the information that they do support.

MXF decoders that implement the Application-Specific Metadata Plug-In mechanism recognize the existence of an extension and can decode its Application Metadata Scheme identifier, the identifiers of the Material Package (extended) Class and its strongly referenced Extension Set Class and, if present, the identifier of the Application Environment for which the information within the extension is intended. From the Primer Pack of the Header Metadata instance, the MXF decoder is also able to determine the unique identifiers of all of the

extension Properties A through E that are encoded in the Application Plug-In Object and Application Referenced Object, respectively.

Even if an MXF decoder that implements the Application-Specific Metadata Plug-In mechanism does not support the extensions, it can pass their information to a higher-level application or can filter it from the MXF file.

MXF decoders that implement the Application-Specific Metadata Plug-In mechanism and its payload can use the information in the file to instantiate the Material Package (extended) and Extension Set Classes.

## **9.8 Descriptive Metadata Plug-Ins**

### **9.8.1 General** (Informative)

This section defines the Descriptive Metadata plug-in mechanism and generic label used to identify the Descriptive Metadata sets.

General Guidelines for adding Descriptive Metadata can be found in the Engineering Guideline for MXF Descriptive Metadata SMPTE EG 42.

### **9.8.2 Generic Universal Label for the MXF Descriptive Metadata Schemes**

The Generic UL below shall be used to identify MXF Descriptive Metadata Schemes.

There may be non-MXF Descriptive Metadata Schemes which use the MXF Descriptive Metadata plug-in mechanism. These Descriptive Metadata Schemes may have different Label and key values to the ones given here.

All MXF Descriptive Metadata Schemes shall be identified with the Universal Label defined below.

Note: This means that all Descriptive Metadata Schemes that are identified by a Label conforming to the definition below are MXF Descriptive Metadata Schemes.

**Table 21 – Generic Universal Label for MXF Descriptive Metadata Schemes**

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

### 9.8.3 Generic MXF Descriptive Metadata Keys

The Generic Key below shall be used for all Metadata Sets that belong to MXF Descriptive Metadata Schemes. Individual Key values shall be defined in the specification of the MXF Descriptive Metadata Scheme.

**Table 22 – Generic Key for MXF Descriptive Metadata Schemes**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets and packs
6	Registry Designator:	xxh	Local Sets: 2-byte Local Tags with either 2-byte lengths (default) or BER encoded length see Section 6.3.5 and appropriate DM Specification
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	vvh	Registry Version in which the specific Scheme and Set first appeared
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	04h	MXF / AAF Descriptive Metadata sets
12	Structure version	01h	Version 1
13	Structure / Scheme Kind	xxh	See scheme for definition
14~16	Reserved	yyh	Reserved for use by each scheme (default 00h)



Note 1: See Section 9.3 for further KLV encoding details. In particular, MXF encoders must use 2 byte local length encoding for all sets in which the values of all Properties have a length equal or smaller than 65535 bytes.

Note 2: According to SMPTE ST 336, the xx has the value of 13h for BER long or short form encoded length and 53h for 2-byte length.

#### 9.8.4 Universal Labels for Abstract Descriptive Metadata Groups

The common Universal Label structure for all Abstract Descriptive MXF Header Metadata Groups in this document shall be defined as follows:

**Table 23 – Common Universal Label Value for all Abstract Descriptive Metadata Groups**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets and packs
6	Registry Designator:	7fh	Abstract Groups, no KLV encoding syntax specified Note: This value used for the same purpose by the revision of ST395 that replaces SMPTE 395M-2003.
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	vvh	Registry Version in which the Key of the specific Group first appeared
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	04h	MXF / AAF Descriptive Metadata sets
12	Structure version	01h	Version 1
13	Structure / Scheme Kind	xxh	See scheme for definition
14~16	Reserved	yyh	Reserved for use by each scheme (default 00h)

The definition of bytes 12, 13, 14 and 15 of the Universal Labels for the two Abstract Descriptive Header Metadata Groups defined in this specification is given in Table 24.

**Table 24 – Universal Label Values for Abstract Descriptive Metadata Groups**

Set Name	Byte 12	Byte 13	Byte 14	Byte 15	Detailed Abstract Group Definition
Descriptive Framework	01h	00h	00h	00h	D.1
Descriptive Object	00h	00h	00h	00h	D.2

Note 1: Abstract Groups are never encoded as Metadata Sets.

Note 2: Other SMPTE standards (such as Descriptive Metadata Schemes) define additional Abstract MXF Metadata Groups. SMPTE ST 395 registers the Universal Labels all Abstract MXF Header Metadata Groups and identifies the SMPTE Standard that defines their semantics and use.

### 9.8.5 Plug-In Mechanism

An MXF file may contain Metadata Sets from zero or more Descriptive Metadata Schemes. These Schemes may or may not be MXF Descriptive Metadata Schemes (i.e. Schemes identified by a Label according to Section 9.8.2 and using Keys according to Section 9.8.3).

A DM Scheme comprises one or more Descriptive Metadata Framework (DM Framework). A DM Framework shall be a Concrete Subclass of Descriptive Framework (see Annex D.1). Each DM Framework shall have one or more metadata Properties. The values of these Properties define the metadata values of the DM Framework instance. One or more of the Properties may constitute Strong or Weak References to one or more Descriptive Metadata Objects (DM Object) that belong to the same DM Framework. A DM Object shall be a Concrete Subclasses of Descriptive Object (see Annex D.2). There shall be no In-File Weak References between different DM Framework instances.

Note 1: Generally, DM Objects are grouped together in a DM Framework for semantic reasons or for encoding reasons (e.g. to avoid nesting of KLV encoding).

DM Segment has an optional Property Descriptive Metadata Application Environment ID that is of type UTF-16 string. If present, the value of this Property shall be the RFC 3986 URI identifier of the specific Application Environment to which the information in the DM Framework instance that is strongly referenced from the DM Segment applies.

An Application Environment may, for example, identify a domain of applications with specific functionality (e.g. broadcast playout) or a specific user application (e.g. the specific production facility of a specific broadcaster).

The inheritance hierarchy of Descriptive Metadata Sets is illustrated in Figure 21. DMS-1 Framework and DMS-1 Set are Superclasses of the DM sets defined in SMPTE ST 380.

Figure 21 also shows the DM Segment Set of the MXF Structural Metadata, which provides the DM Framework Property, which represents the Strong Reference to the DM Framework.

Concrete Subclasses of Descriptive Object may strongly reference other Concrete Subclasses of Descriptive Object.

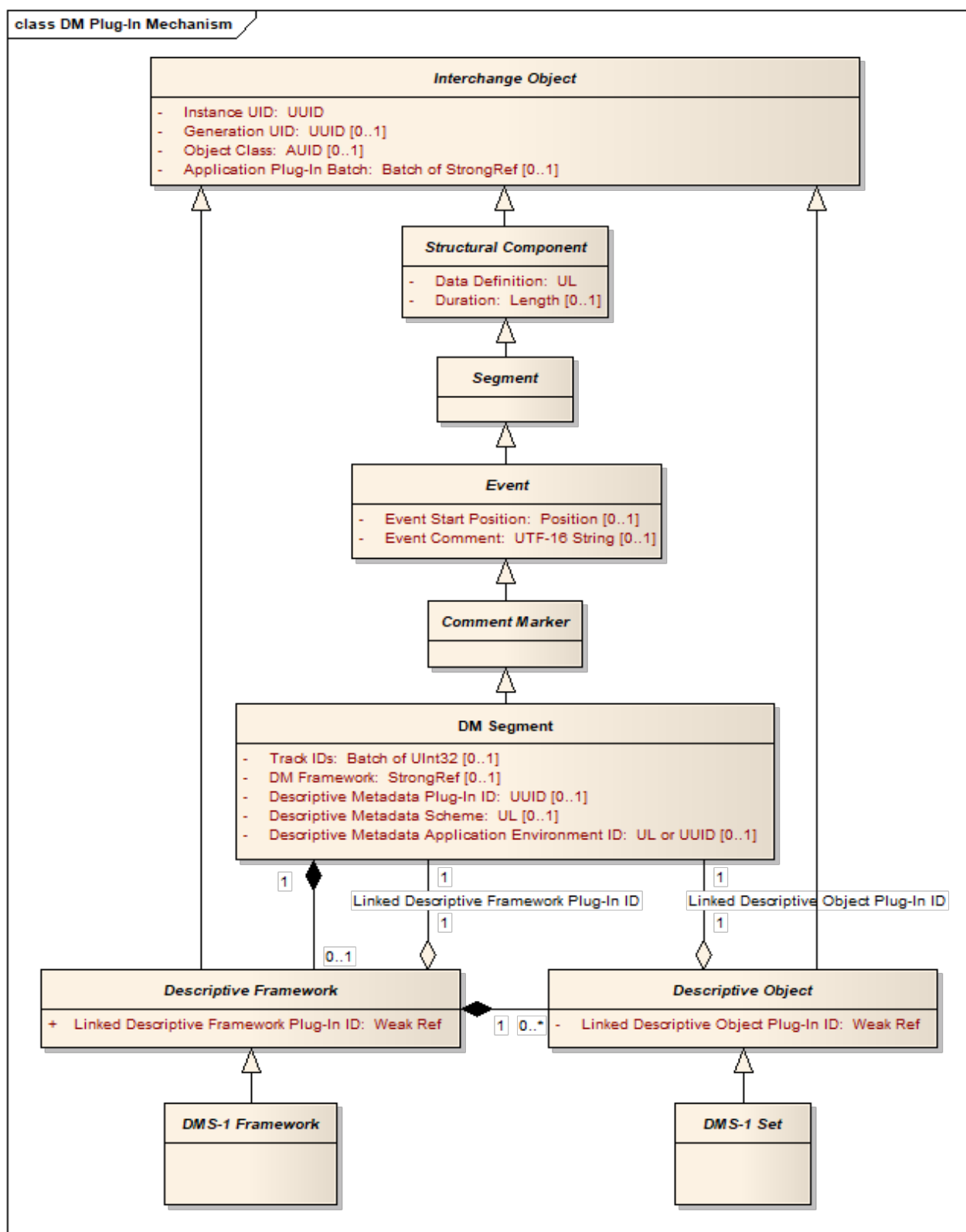


Figure 21 – Descriptive Metadata Plug-In Mechanism

In order to synchronize the Descriptive Metadata to the Essence, the DM Framework shall be strongly referenced (directly or indirectly) from the DM Framework Property of a DM Segment (see Annex B.32). The DM Segment shall be strongly referenced from a Descriptive Metadata Track (DM Track, see Annex B.27) in a Material Package, a Top-Level Source Package or a Lower-Level Source Package.

If the DM Track is in a Material Package then the Descriptive Metadata shall refer to the Essence content as it is to be presented (the generally the output timeline). If the DM Track is in a Top-Level File Package then the metadata shall refer to the Essence content of each Essence Container associated with that File Package (the input streams). If the DM Track is in the Lower-Level Source Package, then the metadata shall refer to the content associated with that Source Package.

Unless specified in a Descriptive Metadata document, a DM Framework may be contained by any kind of Package.

Some DM Frameworks or DM Framework instances will relate to an entire Package. In this case either

1. The duration of the DM Track shall be set to be the duration of the entire Package or
2. The DM Track shall be a Static Track (DM).

Note 2: DM Tracks can be Event Tracks. This means that DM Segments can overlap. It also means that Segments with zero duration are possible. This allows annotation of instantaneous events occurring within the Package.

#### 9.8.6 Simple DM Plug-In Instance Removability

The Descriptive Framework Class (see Annex D.1) and thus all DM Frameworks have an optional Property called Linked Descriptive Framework Plug-In ID. The Descriptive Object Class (see Annex D.2) and thus all DM Sets have an optional Property called Linked Descriptive Object Plug-In ID. If present, these Properties shall constitute Global Weak References to the DM Segment that contains the DM Framework or DM Object instance by direct or indirect Strong Reference. The value of these Weak References shall equal the value of the optional Descriptive Metadata Plug-In ID Property of the DM Segment (see Annex B.32).

Note: The Global Weak Reference to the DM Segment can be resolved within the Header Metadata instance, except in situations where the MXF file has been manipulated by applications that DM Segments, but preserve Header Metadata Sets (DM Frameworks or DM Objects) that are dark to them within the file.

The DM Segment has another optional Property called Descriptive Metadata Scheme. If present, the value of this Property equals the Label in the Descriptive Metadata Schemes Batch of the Preface that is associated with the DM Framework that is strongly referenced from the DM Segment.

If present, the four optional Properties Descriptive Metadata Plug-In ID, Linked Descriptive Framework Plug-In ID, Linked Descriptive Object Plug-In ID and Descriptive Metadata Scheme can be used to enable simple MXF applications to completely remove specific DM Framework instances from the Header Metadata, even if the DM Framework or its DM Objects are Dark to the application.

MXF Encoders that add the optional Descriptive Metadata Plug-In ID, Linked Descriptive Framework Plug-In ID, Linked Descriptive Object Plug-In ID and Descriptive Metadata Scheme Properties to DM Plug-Ins shall obey the following rules:

1. If a DM Segment contains the Descriptive Metadata Plug-In ID Property, the Descriptive Framework and all Descriptive Objects that are contained by the DM Segment by direct or indirect Strong Reference shall contain the Linked Descriptive Framework Plug-In ID and Linked Descriptive Object Plug-In ID Properties, respectively.
2. The value of the Linked Descriptive Framework Plug-In ID and Linked Descriptive Object Plug-In ID Properties of the Descriptive Framework and all Descriptive Objects that are contained by the DM

Segment by direct or indirect Strong Reference shall equal the value of the Descriptive Metadata Plug-In ID Property of the DM Segment that contains them.

3. The value of the Descriptive Metadata Plug-In ID Property shall be unique within a Header Metadata instance. It may remain constant across multiple copies of the Header Metadata.
4. If a DM Segment contains the Descriptive Metadata Plug-In ID Property, it shall also contain the Descriptive Metadata Scheme Property.
5. The value of the Descriptive Metadata Scheme Property shall equal the Label in the DM Schemes Batch of the Preface that is associated with the DM Framework contained by the DM Segment.

In order to enable simple removal of DM Plug-in instances, MXF encoders should use 2-byte Local Tag encoding for Descriptions and add the Descriptive Metadata Plug-In ID Property to all DM Segments.

Note: MXF files generated by MXF encoders compliant to SMPTE 377M-2004 will not have the Descriptive Metadata Plug-In ID Property. MXF decoders compliant to SMPTE 377M-2004 will not recognize the Descriptive Metadata Plug-In ID, Linked Descriptive Framework Plug-In ID, Linked Descriptive Object Plug-In ID and Descriptive Metadata Scheme Properties and will treat them as Dark Metadata.

### 9.8.7 Simple DM Plug-In Instance Removal Implementation (Informative)

In order to remove a DM Framework instance, a simple application would

1. Remove the value of the Descriptive Metadata Scheme Property from the DM Schemes batch of the Preface provided
  - a. there is no other DM Segment with a Descriptive Metadata Scheme Property that shares the same value and
  - b. there is no other DM Framework that shares the same Object identifier (KLV Key) as the DM Framework instance to be removed.
2. Remove all Header Metadata Sets that have a Linked Descriptive Framework Plug-In ID or a Linked Descriptive Object Plug-In ID Property value that equals the Descriptive Metadata Plug-In ID Property of the DM Segment.
3. Remove the DM Segment.

## 9.9 Start Timecode Values

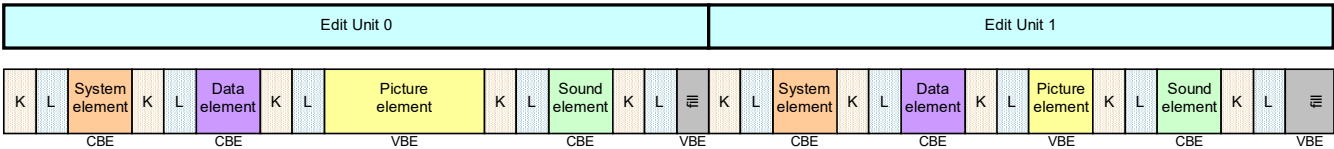
This section provides rules for assigning proper values for the Start Timecode item in the Timecode Component in both Material Packages and Source Packages. Picking the correct value for Start Timecode can be influenced by various metadata values as well as the intended use of the final MXF file. These factors include the presence or absence of timecode values in the System Element in the underlying Essence Container, timecode discontinuities in the Essence Container, the value of the Origin item in the File Package, and whether the Material Package is used to redefine the timecode. The following rules are intended to provide clear, normative constraints based on these factors.

1. For a Source Package Timecode Component in the primary timecode track at offset N, the value of the Start Timecode should be equal to the timecode in ContentPackage[N]. However, if the Essence Container does not include a System Element with a valid Timecode, the Start Timecode value shall be unconstrained.
2. For a Material Package Timecode Component at position N, and given a corresponding Source Package Origin value of M, the value of the Start Timecode should be equal to the timecode in ContentPackage[N+M]. However, since the Material Package can be used to redefine the timecode for playout, the Material Package Start Timecode may have any value.

## 10 File Body

Immediately following the File Header is the File Body, carrying Essence Containers, Index Tables or Header Metadata repetitions. This section defines the MXF Essence Containers. Section 11 defines the Index Tables.

The File Body may carry zero or more Essence Containers. Each Essence Container contains KLV encoded Essence data (which may also include system data and embedded metadata). Each Partition shall contain no Essence Container or one segment of a single Essence Container (see below) which, through its Essence Container specification, defines the KLV encoding. The principle of KLV encapsulation of a typical Essence Container is shown in Figure 22.



**Figure 22 – Typical KLV Coding in an Essence Container wrapped every Edit Unit**

### 10.1 Essence Containers

The individual Essence Container specifications are defined in other documents not included here.

All compliant Essence Container specifications shall meet all the normative criteria listed in Sections 10.2 and 10.3. An Essence Container specification may also define how Index Tables shall be used and how the Essence Descriptor shall be coded. These criteria are listed under two broad headings: the technical requirements and the standardization requirements.

An Essence Container specification may be split over multiple documents when generic structures and specific mappings of different Essence types are required.

### 10.2 Technical Requirements for MXF Essence Containers

An Essence Container shall be coded as a concatenated sequence of individually coded KLV packets where each KLV packet meets the following requirements:

1. Each KLV packet shall be coded according to SMPTE ST 336 and the values of the KLV packet Key shall be publicly registered with the SMPTE.
2. The Essence Container shall be formed by a stream of one or more KLV packets.
3. The Length field of each KLV packet shall be coded according to SMPTE ST 336 with a limit to the coding range provided to limit the requirements of compliant equipment for this Essence Container.
4. The Essence Container specification shall define the byte order. Where multi-byte values are used, the specification shall include the byte order for the correct interpretation of the multi-byte values.
5. The Essence Containers used in streaming Operational Patterns shall be capable of interleave over a defined interleaving period or shall be capable of being multiplexed in an MXF file using the Partition mechanism. The interleave or multiplex duration is dependent upon the application, but should be the period of the minimum duration of usable picture Essence, typically a picture frame period.
6. The KLV packets of each interleave period should contain Essence of essentially the same timing (for example, audio-video timing is rarely sample-accurate). Special timing arrangements may be needed in the case of, for example, long GOP coding using B-frame coding. This requirement allows simple editing and switching of the interleaved Essence.

7. The Essence Container specification shall be assigned a registered SMPTE Universal Label value, which shall be used by the Partition Pack, the Descriptor Set and the Preface Set to identify the Essence Container type.
8. The Essence Container specification shall define which Descriptor Set(s) shall be used to describe the contents of the Essence Container payload.
9. The Essence Container specification may define the use of Descriptor(s) for that Essence Container type.
10. The Essence Container may contain KLV packets containing separate streams of metadata in addition to Essence data.
11. The Essence Container specification should define the use of Index Tables for that Essence Container type.
12. The Essence Container specification may define any mappings required from Multi-track audio in the Header Metadata to Multi-channel audio in the Essence Container.
13. The Essence Container specification shall define the mechanism to create unique Track Numbers for the purpose of identifying specific content within the Essence Container.
14. The Essence Container specification may define a KAG value (see Section 6.4.1).

### 10.3 Standards Requirements of an MXF Essence Container document

An MXF Essence Container specification should be a public standard from an internationally accredited standards body. This is to ensure that all Essence Container documents can be used for interchange.

Note: New Essence Container specifications are recommended to use the MXF Generic Container defined in SMPTE ST 379, where applicable. Otherwise they are recommended to use the Generic Streams mechanism defined in SMPTE ST 410. Doing so will minimize the implementation burden for equipment makers and increase the chances of interoperability.

### 10.4 General Information (Informative)

The size of each element (in bytes) can be determined by the Length value of the KLV packet according to SMPTE ST 336 in conjunction with the defining Essence Container specification.

The Essence Container specification could comprise more than one document. This is the case with the Generic Container which requires associated Essence and metadata mapping documents together with application documents to provide a complete specification.

The interleaving of elements of different Essence Types is fully described in each Essence Container document.

The description of each Essence type is defined in the appropriate Essence Container Specification.

### 10.5 Descriptors

The Header Metadata identifies the Essence Container and describes its Contents(s) such that Packages are concerned with temporal characteristics and Descriptors are concerned with parametric Properties. These Properties could be related to sampling, such as sampled screen size. They could also be related to the organization of the data such as the number of audio channels. They could also provide further information such as where external Essence may be located.

All Descriptors shall be derived from the Generic Descriptor which is defined in Annex B.2. It is a generic Class that defines the base functionality of all Descriptors. It is never used directly (i.e. it is an Abstract Superclass). For each individual Descriptor that is derived directly or indirectly from the Generic Descriptor, Item Designator, Set Keys and Properties shall be defined.

### 10.5.1 Use of Descriptors in File Packages

File Packages describe Essence that is internal Essence in Essence Containers within the MXF file or external Essence in files identified by Network Locators.

When a File Package describes both internal Essence and external Essence, the Essence shall be identical. If a File Package references multiple external Essence files, the Essence of all external files shall be identical.

The Descriptor of a File Package contains detailed information about the Essence format(s) of the associated Essence Container. The values of the Properties of the Descriptor shall conform to the Essence in the associated Essence Container. The values of the Properties of the Descriptor shall conform to the Essence in files referred to by Network Locators of the File Package.

Note 1: The external file format can contain additional data as a preamble to the Essence data; an optional Property is provided in the Index Table to indicate the offset in the external file where the Essence data starts.

When a File Package describes internal Essence, the correct value of the Essence Container Property shall be the Universal Label that identifies the Essence Container used within the MXF file (for example, the specific kind of Generic Container used for picture or sound data as listed in an Essence mapping specification).

When a File Package describes external Essence and not internal Essence, the correct value of the Essence Container Property of the Descriptor should be the Universal Label that identifies the external file format (for example, RIFF WAVE, which is commonly used as a container for PCM data). The SMPTE registry that constitutes the normative reference for these values is defined by SMPTE ST 400. The values are listed in SMPTE RP 224.

If the external file format is not known, the correct value of the Essence Container Property of the Descriptor shall be the Universal Label that identifies the file format as unknown (i.e. the Label 06.0E.2B.34.04.01.01.0A.01.01.02.01.01.00.00.00).

If the File Package references multiple external Essence files, and if not all of those files are of the same file format, the correct value of the Essence Container Property of the Descriptor shall be the Universal Label that identifies the file format as unknown.

Note 2: Refer to Section 6.2.3 for the provisions when a required Property must contain its correct value.

When a File Package describes both internal Essence and external Essence, or when the File Package describes external Essence and the value of the Essence Container Property of the Descriptor equals the Universal Label that identifies the file format as unknown files, the kind of external file is not identified by the Essence Container Property of the Descriptor, and must be discovered by inspection of the external file.

### 10.5.2 Use of Descriptors in Physical Packages

Physical Packages shall describe Essence that is external, and is on media identified by Network Locators or physical media identified by Text Locators.

The Descriptors in Physical Packages should contain basic information about the media and the Essence. The media and the Essence on the media referred to by Locators of the Physical Package shall conform to the values of the Properties of the Physical Descriptor.

If a Physical Package simultaneously references multiple external Essence instances, the Essence of all external Essence instances shall be identical.

Note: This standard does not define any physical descriptors.



### 10.5.3 Use of Locators

The Locators Property of the Generic Descriptor is an Array of Strong References to Locator sets. The Locator sets hold location information for external Essence.

The Essence may not exist at the specified location. MXF applications shall be able to recognize that Essence is external to the file through the presence of one or more Locators. MXF applications that support external Essence should have a defined behavior (such as reporting an error) if the external Essence at the specified locations cannot be accessed.

When multiple locations are specified (using multiple Locator sets), MXF decoders shall search the external Essence in the order defined by the Array of Strong References.

Locators shall only be present in Descriptors that are referenced directly from a Source Package. They shall not be present in Descriptors that are referenced from a Multiple Descriptor.

### 10.5.4 Extending Essence Descriptors

New MXF mapping documents may provide extensions to existing Generic Descriptors by one of the following three methods:

1. Creation of a new Subclass of an existing Generic Descriptor,
2. Addition of new optional Properties to an existing Generic Descriptor,
3. Addition of one or more SubDescriptors to existing Generic Descriptors.

Method 1 shall involve taking all Properties of the existing Generic Descriptor, adding one or more new Property. It shall involve the definition of a new KLV key for the new Descriptor. It shall be used when the extension adds one or more Req, E/req or D/req Property and when all of the new Properties apply to only one already defined Descriptor.

Method 2 shall involve taking all Properties of the existing Generic Descriptor, adding one or more new optional Properties. It shall not result in the assignment of a new KLV key. It shall only be used when all of the new Properties apply to only one already defined Descriptor.

Method 3 involves defining a SubDescriptor Set and specifying the Generic Descriptors that shall be able to reference it through the SubDescriptors Property. It shall be used when the extension applies to more than one already defined Descriptor.

The Superclass of all SubDescriptors is defined in Annex B.3.

The SubDescriptors Property shall reference one or more SubDescriptor sets. MXF encoders shall not encode a SubDescriptors Property that references zero SubDescriptors sets.

Note: The method of adding a Sub-Descriptor was first used in SMPTE ST 422.

### 10.6 Interleaved Essence Containers

An interleaved Essence Container shall be an Essence Container that combines two or more Essence Elements that conform to the same or different Essence Container specifications such that the Essence Container byte stream consists of a sequence of two or more Essence Element groupings that consist of KLV Packets containing Essence data from each Essence Element.

Essence Container specifications may define additional constraints that apply to this situation.

SMPTE ST 379 requires that, within an interleaved Essence Container, the number of KLV Packets within in an Essence Element grouping (e.g. an Edit Unit of stored Essence) must remain constant for all Essence Element groupings. This provision of SMPTE ST 379 assures that it is possible to determine the order of Essence Elements in all groupings of an interleaved Essence Container by only inspecting the first (or any other) Essence Element grouping.

Example: For complex interleaves of compressed audio (see Section 11.1.3), this means that, if more than one Sound frame is required for an Essence Element within an Essence Element grouping of a interleaved Essence Container, then all Sound frames are assembled within a single KLV package. If no Sound frame is available for an Essence Element, then a placeholder Essence Element is inserted that uses the same KLV key as the Sound Element and has KLV length value of zero (0).

## 11 Index Table

### 11.1 Overview

Index Tables can be used to speed up the location of individual Edit Units of stored Essence in a timeline. The Essence itself may be interleaved, as in the Generic Container, or may be a single Essence stream.

An Index Table may be placed in the Header Partition, in a Body Partition, in the Footer Partition or distributed over several Partitions of an MXF file (see Section 6.2.1).

Index Tables can be very simple and lightweight for Essence that has a constant number of bytes per frame, yet can also support complex Essence that has a varying bit-rate and re-ordered frames such as can be encountered with predicatively compressed Essence.

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

An Index Table can be created ‘on the fly’ during file creation from an input signal and is notionally placed in the File Footer on recording. In practice, its placement in a server file system may be anywhere for storage convenience. During transfer of a complete file, an application may choose to replace a distributed Index Table with one placed immediately after the Header.

Note 1: Streaming devices can have limited Index Table memories and, in this circumstance, a large concentrated Index Table is likely to be less useful than a distributed one.

Index values and offsets do not change if the Index Table is relocated within the file – this is handled by the low-level byte-stream format described in Section 7, in which the Index Table and Essence Container are treated as separate Streams linked together by the values of Properties in the Partition Pack.

When indexing Essence within an MXF file, all Index Table Entries use byte offsets relative to the start of the Essence Container Stream, and not absolute offsets within the file. When indexing external Essence in non-MXF files, the stream offset shall always be equal to the absolute byte offset from the start of the first Essence byte in the external file.

An Index Table shall be used to index a single Essence Container. Each Index Table shall index Edit Units stored Essence of the Essence Container. The Edit Unit rate of an Index Table is defined by the Edit Rate of the Essence Tracks of the Package that describes the Essence Container that the Index Table indexes.

Note 2: An Edit Unit is usually the time period of a field or a frame of video. An Edit Unit for an audio-only application could be the time period of an AES3 audio frame comprising 192 audio samples (having duration of 4 ms when using a 48-kHz sampling rate).

The Index Tables are specified in such a way that they can provide indexing support for Picture, Sound, field-coded, frame-coded, interleaved and variable bit rate Essence. In the section below, the word “picture” will be used to refer to an Essence image which has been field or frame coded and may be interleaved with other Essence such as sound or data.

The Index Tables are also able to cope with temporally re-ordered content such as Long GOP MPEG. Each of the Essence Container specifications may define implementation details of Index Tables for that particular Essence Container. The examples below are provided to aid the designer build a generic Index Table handler for applications ranging from non-interleaved simple Essence to interleaved and temporally reordered Essence.

Note 3: Application writers need to be aware that absence of Index Tables could lead to performance degradation, especially when MXF files are transferred to and from certain classes of machines. Application performance could also be degraded by the requirements of an Index Table implementation. Their use is therefore optional.

### 11.1.1 Interleaved Streams

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

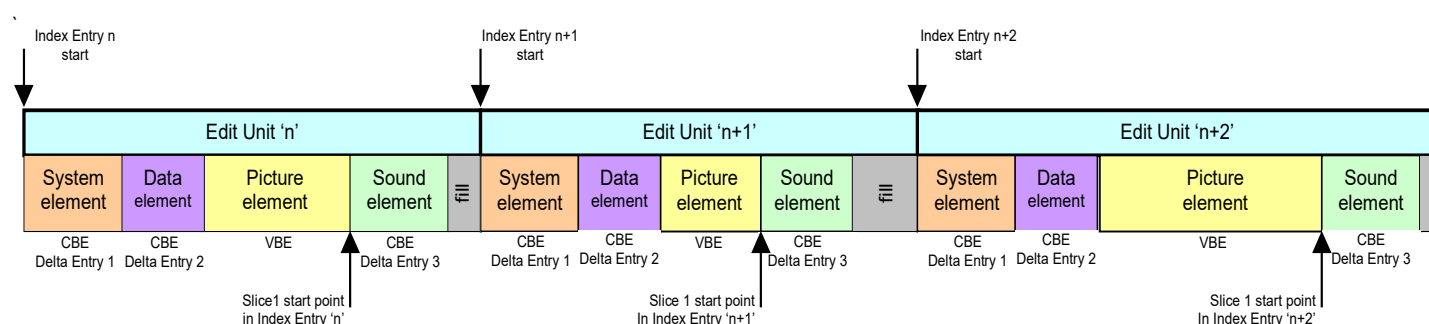
Note: Further details can be found in SMPTE ST 379, the specification of the Generic Container.

### 11.1.2 Constant Bytes per Element (CBE) and Variable Bytes per Element (VBE) (Informative)

Within an interleaved Essence Container, each Essence Element is either a Constant Bytes per Element (CBE) or a Variable Bytes per Element (VBE).

Whenever there is at least one VBE in an Essence Element grouping, a separate Index Table value could be required to locate each subsequent Essence Element in the interleaved stream. Alternatively, a Fill Item could be added to each VBE Element to pad it out to an equivalent CBE boundary.

Note 1: An uncompressed audio Essence Element has the constant bytes per Element property unless the video field rates lead to small variations (e.g. at a 59.94-Hz field or frame rate, the number of audio samples varies between 800 and 801 in a regular sequence).



**Figure 23 – Index Table relationships for Interleaved CBE/VBE Stream**

Figure 23 shows the first three Edit Units of stored Essence in an interleaved Essence Container (which was KLV coded in Figure 22). In this example, each interleaved Essence Element grouping consists of a system, data, picture and sound Element. The system, data and sound Elements are CBE, while the picture Element is VBE. The figure shows the concept of an Index Table Slice. An Index Table Slice consists of zero or more CBE Elements followed by a VBE Element or the end of the Edit Unit of stored Essence.

Note 2: The term ‘slice’ as used above must not be confused with the same term as used by ISO/IEC 13818-2 (MPEG2).

11.1.3 Complex Interleaves of Compressed Audio

There are certain conditions in which the distribution of Sound and Data Elements can be uneven. This can occur, for example, when compressed Sound with large, indivisible frames is interleaved with Picture having different frame duration. Under this scenario, the following conditions can arise:

- 1. If the frequency of the Sound frames is less than that of the Picture frames, there could be Essence Element groupings for which there is no Sound frame.
- 2. If the frequency of the Sound frames is greater than that of the Picture frames, there could be Essence Element groupings for which there are more than one Sound frame.
- 3. Within an Essence Element grouping, the temporal location of the start of the first Sound frame may not be synchronized with the start of the first Picture frame.

Note 1: The rules defined in Section 10.6 specify how the Sound frames need to be packed into KLV items of the Essence Container such that it is possible to determine the order of Essence Elements in all groupings of the interleaved Essence Container by only inspecting the first (or any other) Essence Element grouping.

In order to correctly align the synchronization points of the Picture and Sound, it is required to know the relative temporal offsets among the Essence Element payloads in the indexed Edit Unit.

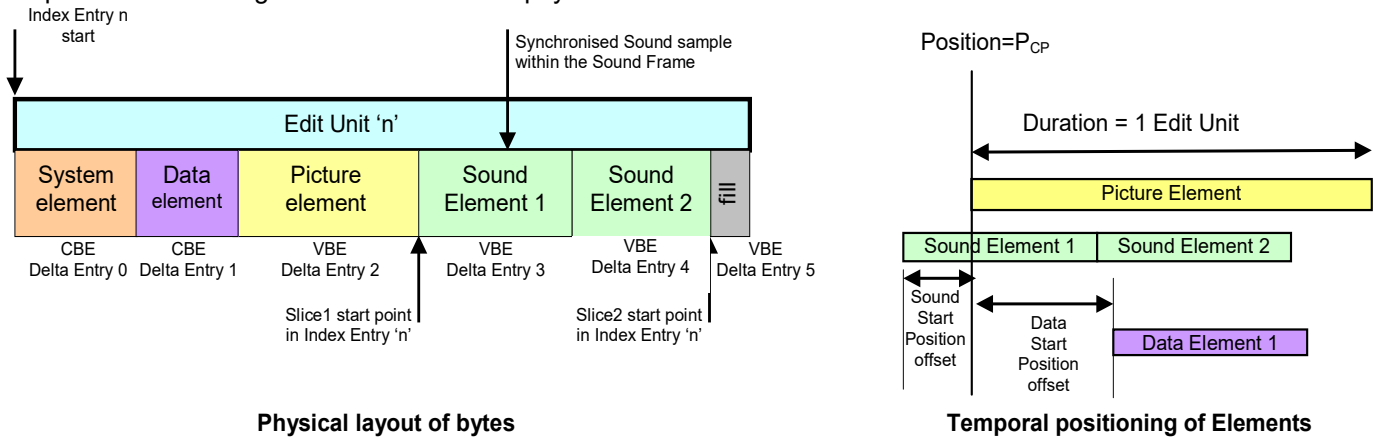


Figure 24 – A Content Package with Interleaved Large Sound Frames

The figure above shows an Essence Element grouping containing 5 Essence Elements and a Fill Item. The interleaving of the Data, Picture and Sound can either lead to a different number of Sound frames in the Sound Element in the same Edit Unit of stored Essence and therefore turns it into a VBE Element (see Section 10.6 for further information).

The dark arrow at the top of Figure 24 indicates the Sound sample within the indivisible Sound frame which is synchronized with the Picture Element payload. A temporal offset is recorded in the Index Table as the temporal displacement between starts of the Picture Element and Sound Element payloads (e.g. the first Sound sample in the Sound Element). The temporal displacement is expressed as a fraction of the duration of the Edit Unit. This fractional measure allows precise synchronization of Picture, Audio and Data Element payloads.

In Figure 24, the Position Table entry (Pcp) that corresponds to the Sound Element specifies a negative Sound Start Position offset (as a negative fractional value of the duration of the Edit Unit) and the Position Table entry

that corresponds to the Data Element specifies a positive Data Start Position offset (as a positive fractional value of the duration of the Edit Unit).

Note 2: This mechanism is optional. Quantizing the synchronized components to a single frame is sufficient in many applications.

#### 11.1.4 Description of Operation

An Index Table provides byte offset information within an Essence Container for a given time offset from the start of that Essence Container. If the Essence Container has interleaved data within it, then extra mechanisms are provided for finding the offsets to the individual Essence Elements once the correct time offset is located. Each Index Entry provides the byte offset within the Essence Container or the indexed external file for a given time offset measured in Edit Units. To locate the individual Essence Elements within the Index Entry, the Delta Entries and possible Slice Offsets are required. The extent to which the Essence is indexed depends on an application. For many applications, simple indexing of the start of each Edit Unit will suffice. It is then up to the decoder within the application to find the start of each Essence Element by parsing the stored Essence.

##### Index Entries

An Index Entry has one Stream Offset value, zero or more Slice Offset values and zero or more Delta Entry values.

**Stream Offset**  
Provides the absolute byte offset to the start of the indexed Edit Unit within the Essence Container or the indexed external Essence file, such that

1. For frame-wrapped Essence, each Index Entry value shall mark the start of the Key for the KLV packet of the first Essence Element in the Edit Unit of stored Essence.
2. For clip-wrapped Essence, each Index Entry value shall mark the start of an Edit Unit of stored Essence, excluding the KL at the start of the Essence Container in each Partition.

Note 1: Including the bytes occupied by KL at the start of the Essence Container would eliminate the ability to use a single Index Entry to index CBE Essence.

3. For custom-wrapped Essence that uses superframe-wrapping, each Index Entry value shall mark the start of an Edit Unit of stored Essence, excluding the KL at the start of the superframe-wrapped Content Package.
4. For custom-wrapped Essence that uses subframe-wrapping, each Index Entry value shall mark the start of the Key for the first KLV packet of the first Essence Element in the Edit Unit of stored Essence.

Note 2: The terms frame-wrapped, clip-wrapped, custom-wrapped, superframe-wrapped, subframe-wrapped and Content Package are defined in SMPTE ST 379.

The temporal distance between Index Entries within an Index Table Segment Set shall be one Edit Unit.

##### Slice Offsets

Each Index Entry value may have zero or more Slice Offset values that provide the byte offset within the Edit Unit of stored Essence to the end of any Essence Elements which are VBE.

Each Slice shall start with zero or more indexed CBE Elements and shall end with a single VBE Element or the end of the Edit Unit of stored Essence.

If the arrangement of Essence Elements in an Essence Container is such that the only VBE is the last Essence Element, there should be no Slices in the Index Table Segments.

The start of Slice zero shall correspond to the start of the Edit Unit of stored Essence.

## Delta Entries

A single Delta Entry Array provides information to find the Essence Elements within an interleaved Essence Container.

There shall be one Delta Entry for every indexed Essence Element within the Essence Container. However, some Essence Elements within the Essence Container may not be indexed.

Any Fill item may be indexed if desired.

Example: If all Essence Elements of a 3 Element interleave in the order CBE, VBE, CBE are indexed, the Index Table Segment must have the following 3 Delta Entries:

CBE Element:      Delta Entry (slice number=0, delta from start of slice= 0)

VBE Element:      Delta    Entry    (slice    number    =0,    delta    from    start    of    slice=  
sizeof(first\_CBE\_element))

CBE Element:      Delta Entry (slice number =1, delta from start of slice= 0)

Any Element that has minor sample variations (e.g. the 800/801 525-line audio sequence) shall be padded to a constant size if it is to be regarded as a CBE. Otherwise it shall be a VBE Element and the Slice mechanism shall be used for indexing the next indexed Essence Element.

An Index Entry does not provide information about any interleave within the Essence Container. The Essence Type of the Delta Entry shall be determined by inspection of the Key of the Essence (e.g. System, Picture, Sound, Compound, Fill etc.).

An Index Entry does not specify which Essence Elements of an interleaved Essence Container are indexed using Slice Offsets or Delta Entries. MXF decoders that use the Index Table shall determine this by inspecting the Essence Element Keys in an Edit Unit of stored Essence and by comparing their offsets to the ones specified in the corresponding Index Entry. When the offset of an Essence Element in the Edit Unit of stored Essence equals an offset in the Index Entry, then the Essence Element shall be assumed to be indexed. Otherwise it shall be treated as not indexed.

Note 1: If an Essence Element is not indexed, its offset can be computed from the offset of the previous indexed Essence Element and the KLV lengths of this and, possibly, subsequent non-indexed Essence Elements up to the desired Essence Element.

Throughout the entire duration of the Essence Container, the same delta entry of all Index Table Segments shall correspond to the same Essence Element or Fill Item.

Note 2: Together with the provision of SMPTE ST 379 that the sequence of Essence Elements within Content Packages of the Generic Container shall remain constant for all Content Packages, this assures that the relationship between Delta entries and indexed Essence Elements can be determined by only inspecting the Delta Entry Array, one Edit Unit of stored Essence of the Essence Container and its associated Index Entry.

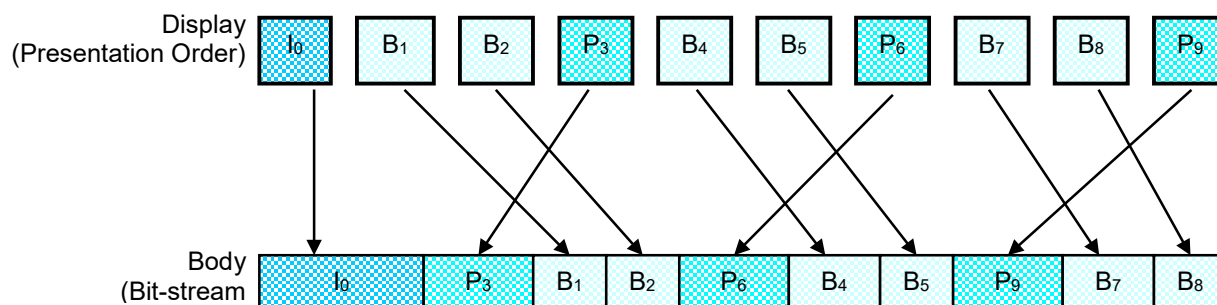
### 11.1.5 Generalization using Element Date

For each Essence Element within the interleaved Essence Container, the Element Delta defines whether the Essence Element is reordered temporally, which Slice contains the Essence Element, and the (fixed) byte offset from the start of the Slice to the start of the Essence Element. In addition, the Delta Entry allows the temporal synchronization of an Essence Element relative to other Elements to be calculated.

There is only one Delta Entry Array per Index Table Segment (see Section 11.2.3), Essence Elements which only appear occasionally in a stream (e.g. a DVB subtitle PES stream) shall be indexed as VBE Element.

### 11.1.6 Temporal Re-ordering

In the particular case of MPEG long-GOP video, the compressed video pictures may be reordered from their display order according to the MPEG specification. This reordering is applied only to the video Elements. An example is shown in Figure 25.



**Figure 25 – Temporal Reordering of MPEG-2 Frame coded Video**

MXF Index Tables provide a rapid conversion from the Position of an Edit Unit into byte offsets within the Essence Container or external Essence file including display order offsets (“temporal offsets”) for long GOP MPEG elements.

Note: The processes defined in this section can be equally applied to any temporally re-ordered Essence stream and are not limited solely to MPEG long GOP streams.

As an example, consider the sequence in Figure 25 and Figure 26. This is a typical frame coded Long GOP MPEG sequence with two B frames. The following example will access field 3, the second field of the first B frame.

The first step of the process is to look up the field number (in display order). From this, the reordering temporal offset of this field is found. This value is added to the original field number and then used as a 2<sup>nd</sup> lookup in the same table to find the Picture type and Offset to the Essence Element within the stream. Note that the Offset to Element is calculated from the Delta Entry and Index Entry information as shown below.

This is represented schematically as follows:

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

**Figure 26 – Conversion from Field Number to Offset with Temporal Offset**

In the case of non-reordered Essence Elements such as Sound, the Temporal Offset in the Index Entry for that Edit Unit is simply ignored when calculating the Offset to Element from the Delta Entry and Index Entry information. This is illustrated in Figure 27.

Field Number	Temporal Offset	Offset to Element
0 (I)	0	00000000h
1 (I)	0	00000000h
2 (B)	+2	0000c250h
3 (B)	+2	0000c250h
4 (B)	+2	0011b240h
5 (B)	+2	0011b240h
6 (P)	-4	0014df80h
7 (P)	-4	0014df80h
8 (B)	+2	0017fe40h

Figure 27 – Conversion from Field Number to Offset ignoring Temporal Offset

11.1.7 Indexing Empty Essence Elements

For indexing purposes, Essence Elements shall be treated the same, whether they are empty or not.

Note: According to SMPTE ST 379 and Section 9.5.5, empty Essence Elements need to be added in some circumstances in order to archive identical values of the Origin for all Essence Tracks of a Top-Level File Package

11.1.8 Indexing KLV Fill Items

KLV Fill Items may be used to pad the Essence Elements to a given grid size. Where present, the KLV Fill Items may have their own entry in the appropriate part of the Index Table (either as a Delta Entry or as a Delta Entry plus a Slice Offset).

11.1.9 Constant Edit Unit Size

In Essence Containers where either

- 1. the size of all Edit Units of stored Essence is constant and all indexed Essence Elements have the same relative offset from the beginning of the Edit Unit of stored Essence or
- 2. item 1 applies for all but the first Edit Unit of stored Essence of the Essence Container,

the Index Entry Array may be omitted.

If condition 1 applies and the Index Entry Array is omitted, the value of the Edit Unit Byte Count Property shall be equal to the size of an Edit Unit of stored Essence and Index Duration shall either be set to 0 or to the total number of Edit Units in the Essence Container. The Essence Container byte offset for the start of Edit Unit of stored Essence N can then be computed as N \* Edit Unit Byte Count, where N=0 is the beginning of the sequence.

If condition 2 applies, and if the Index Entry Array is omitted, there shall be two Index Table Segments that are multiplexed into the same Partition. In the first Index Table Segment, the value of the Edit Unit Byte Count



Property shall equal the size of the first Edit Unit of stored Essence of the Essence Container and the value of Index Duration shall equal 1. In the second Index Table Segment, the value of the Edit Unit Byte Count Property shall equal the size of the second Edit Unit of stored Essence of the Essence Container, the value of Index Start Position shall equal 1 and the value of Index Duration shall either be set to 0 or to the total number of Edit Units in the Essence Container – 1. The Essence Container byte offset for the start of Edit Unit  $N > 0$  can then be computed as Edit Unit Byte Count of the first Index Table Segment +  $(N - 1) * \text{Edit Unit Byte Count of the second Index Table Segment}$ , where  $N=0$  is the beginning of the sequence.

Note: This is the only case where information from more than one Index Table Segment is needed in order to compute the Essence Container byte offset for an Edit Unit.

In Essence Containers where the size of the Edit Units of stored Essence is constant, and the Index Table only indexes the start of the Edit Units, the Delta Entry Array may be omitted.

The individual Essence Container specifications shall identify if the Edit Unit of stored Essence and its Essence Elements are of constant size and, if this is the case, define appropriate values. Index Table Segments with these values may appear in any relevant Partition, but each appearance of the Index Table Segment shall have identical values.

A value of zero in the Edit Unit Byte Offset item shall identify that the size of the Essence Elements in each Edit Unit of stored Essence is not constant and that the indexing of Edit Units is to be found in the Index Entry Array.

## 11.2 Index Table Specification

An Index Table is intended to provide a conversion between Edit Units and byte offsets within an Essence Container in the MXF file or in an external Essence file. In an interleaved Essence Container or external Essence file, there may be several different components such as picture, sound and data. Each of these components will have an associated Track in the Package which describes it.

Note 1: In order to construct a valid Index Table, the Edit Rate Property of each indexed Track in the Package that describes the indexed Essence Container has the same value (see Section 9.5.5, item 12).

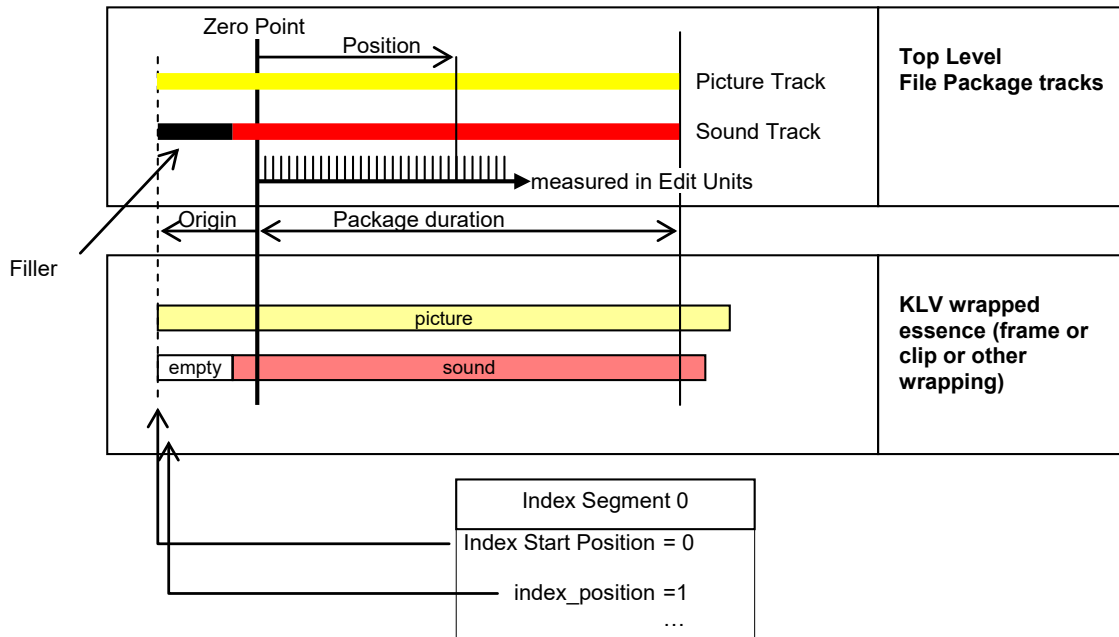


Figure 28 – Index Table timing model

Essence is indexed by the time offset from the start of Essence as defined in Section 9.4.2. The value of *index\_position* shall be used as a lookup in an index table segment to obtain the byte offset of an Essence Element within the Essence Container.

Index Tables are specified as Index Table Segments. A complete Index Table shall comprise one or more Index Table Segments.

If the size of the Index Entry Array exceeds 65,535 bytes, the Index Table should be segmented such that its individual Index Table Segments can be encoded using 2 byte length KLV Set encoding.

Note 2: If such an Index Table is not segmented, SMPTE 377M-2004 compliant MXF decoders might not be able to decode it.

### 11.2.1 Index Table Segments

The size of individual Index Table Segments is specific to the application.

Example: The size of Index Table Segments including optional Fill Items could be chosen to be a multiple of the KAG size.

Every Index Table Segment shall specify the identifier of the Index Table (IndexSID) of which it is part, and the identifier of the Essence Container (BodySID) that it indexes.

Zero or more Index Table Segments may be inserted into any Partition according to the rules in Section 6.2.

Index Tables may be repeated in the file. The manner in which this is done is application specific; however, the following rules shall apply:

1. A repeated Index Table Segment may accumulate prior Index Table Segments in the File.
2. When Complete Index Tables are used, they should be placed in the Header Partition and / or the Footer Partition.
3. A Complete Index Table may be divided in order to provide distributed Index Table Segments throughout the Partitions in the file.
4. MXF Encoders shall ensure that for each repeated Edit Unit offset within the file, the byte offset values for each repeated Indexed Essence Element shall be identical.
5. When creating Index Table Segments, the Start Position of consecutive Index Table Segments shall increase such that adjacent Index Table Segments do not overlap.
6. If repetition of the Index Table is required, then the previous Index Table shall be completed before the repetition starts. A repetition of an Index Table shall commence in a new Partition.

### 11.2.2 Index Table Segment Key

A 16-byte SMPTE Universal Label shall be used to identify an Index Table Segment and act as a Key for KLV coding of the Index Table Segment.

The Index Table Segment Key shall have the value defined in Table 25.

**Table 25 – Specification of the Index Table Segment Key**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets and Packs
6	Registry Designator	xxh	Local Sets: 2 byte tags with either 2 byte length (default) or BER coded length
7	Structure Designator	01h	Set/Pack registry
8	Version Number	01h	Registry Version in which the specific Key first appeared
9	Item Designator	0Dh	Organizationally registered
10	Organization	01h	AAF Association
11	Application	02h	MXF File Structure
12	Structure Version	01h	Version 1
13	Structure Kind	01h	File Structure sets and packs
14	Set/Pack Kind	10h	Index Table Segment
15	Version	01h	Index Table Specification version
16	Reserved	00h	

MXF encoders shall use 2-byte local length encoding for all Index Table Segments in which the values of all Properties have a length equal or smaller than 65535 bytes.

MXF encoders shall use BER local length encoding for all Index Table Segments in which the value of one or more Properties have a length of greater than 65535 bytes.

Note: According to SMPTE ST 336, the placeholder value xxh for byte 6 has the value of 13h for BER long or short form encoded length and 53h for 2-byte length.

### 11.2.3 Index Table Segment

The Index Table Segment shall be defined by Table 26.

**Table 26 – Index Table Segment Set**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Index Table Segment	Set Key	16		Table 25	Req	An Index Table Segment Set	
Length	BER Length	var			Req	Set Length (see 9.3)	
Instance UID	UUID	16	3C.0A	06.0E.2B.34 01.01.01.01 01.01.15.02 00.00.00.00	Req	Unique ID of this instance <b>[Note: SMPTE RP 210 definition</b> The ISO/IEC 11578 (Annex A) 16-byte Globally Unique Identifier]	
Index Edit Rate	Rational	8	3F.0B	06.0E.2B.34 01.01.01.05 05.30.04.06 00.00.00.00	Req	Edit Rate copied from the Essence Tracks of the Essence Container <b>[Note: SMPTE RP 210 definition</b> Specifies the indexing rate in hertz]	
Index Start Position	Position	8	3F.0C	06.0E.2B.34 01.01.01.05 07.02.01.03 01.0A.00.00	Req	The first Edit Unit indexed by this Index Table Segment measured in Top-Level File Package Edit Units relative to the first Edit Unit in the Essence Container <b>[Note: SMPTE RP 210 definition</b> Specifies the position relative to start of essence, in edit units, where indexing starts]	
Index Duration	Length	8	3F.0D	06.0E.2B.34 01.01.01.05 07.02.02.01 01.02.00.00	Req	Time duration of this Index Table Segment measured in Edit Units of the referenced Package <b>[Note: SMPTE RP 210 definition</b> Specifies the duration of an Index table in content units]	
Edit Unit Byte Count	UInt32	4	3F.05	06.0E.2B.34 01.01.01.04 04.06.02.01 00.00.00.00	D/Req	Defines the byte count of each and every Edit Unit of stored Essence indexed by this Index Table Segment. A value of zero (0) defines the byte count of Edit Units of stored Essence is only given in the Index Entry Array. <b>[Note: SMPTE RP 210 definition</b> The length of an edit unit (in Bytes) in the container]	0
IndexSID	UInt32	4	3F.06	06.0E.2B.34 01.01.01.04 01.03.04.05 00.00.00.00	D/Req	Stream Identifier (SID) of Index Table According to 11.2.1, this Property must be present in all valid MXF files that use Index Tables. <b>[Note: SMPTE RP 210 definition</b> Index table stream ID]	
BodySID	UInt32	4	3F.07	06.0E.2B.34 01.01.01.04 01.03.04.04 00.00.00.00	Req	Stream Identifier (SID) of the indexed Essence Container <b>[Note: SMPTE RP 210 definition</b> Essence (or its container) stream ID]	
Slice Count	UInt8	1	3F.08	06.0E.2B.34 01.01.01.04 04.04.04.01 01.00.00.00	D/Req	Number of Slices minus 1 (NSL) <b>[Note: SMPTE RP 210 definition</b> Number of sections indexed, per edit unit, minus one]	0
PosTableCount	UInt8	1	3F.0E	06.0E.2B.34 01.01.01.05	Opt	Number of PosTable Entries minus 1 (NPE) <b>[Note: SMPTE RP 210 definition</b> Number of position	0

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
				04.04.04.01 07.00.00.00		offsets indexed, per edit unit, minus one]	
Delta Entry Array	Array of DeltaEntry	var	3F.09	06.0E.2B.34 01.01.01.05 04.04.04.01 06.00.00.00	Opt	Map Essence Elements onto Slices (see Table 27)  [Note: SMPTE RP 210 definition Array of values used to identify elements of Essence within an edit unit]	
Index Entry Array	Array of IndexEntry	var	3F.0A	06.0E.2B.34 01.01.01.05 04.04.04.02 05.00.00.00	D/Req	Index from Edit Unit number to stream offset (see Table 28)  [Note: SMPTE RP 210 definition Array of values used to index elements from edit unit to edit unit]	
ExtStartOffset	UInt64	8	3F.0F	06.0E.2B.34 01.01.01.0A 04.06.02.04 00.00.00.00	D/Req	The byte offset to the first Essence data in an external Essence file  [Note: SMPTE RP 210 definition The byte offset to the first essence data in a CBE (Constant Bytes per Element) essence stream]	0
VBEByteCount	UInt64	8	3F.10	06.0E.2B.34 01.01.01.0A 04.06.02.05 00.00.00.00	D/Req	The count of bytes of the last essence element in the last Edit Unit indexed by the Index Table Segment.  [Note: SMPTE RP 210 definition The byte offset to the end of the final essence unit in a VBE (Variable Bytes per Element) essence stream. Used to calculate the size of the final essence unit.]	0
Single Index Location	Boolean	1	3F.11	06.0E.2B.34 01.01.01.0E 04.04.05.01 00.00.00.00	Opt	If all Index Table Segments that compose one Complete Index Table are in one Partition, the Boolean value shall be TRUE. Else, (Index Table Segments that compose one Complete Index Table are distributed into multiple Partitions), the Boolean value shall be False.  [Note: SMPTE RP 210 definition Specifies whether the Index Table Segments are in one partition or multiple partitions.]	
Single Essence Location	Boolean	1	3F.12	06.0E.2B.34 01.01.01.0E 04.06.02.06 00.00.00.00	Opt	If all Essence Containers are in one Partition, the Boolean value shall be TRUE. Else, (the Essence Container Segments are placed into multiple Partitions), the Boolean value shall be FALSE.  [Note: SMPTE RP 210 definition Specifies whether the Essence Containers are in one partition or multiple partitions.]	
Forward Index Direction	Boolean	1	3F.13	06.0E.2B.34 01.01.01.0E 04.04.05.02 00.00.00.00	Opt	If all Index Table Segments that compose one Complete Index Table precede Essence Container Segments that they index, then the Boolean value shall be TRUE. Else (all Index Table Segments that compose one Complete Index Table follow Essence Container Segments that they index), the Boolean value shall be FALSE.  [Note: SMPTE RP 210 definition Specifies whether the Index Table Segments are pointing forward or backward.]	

Conditions:

1. Where the Edit Unit Byte Count value is non-zero, the Duration value may be set to zero to indicate that this Index Table Segment applies to the all Edit Units of stored Essence in the Essence Container identified by this BodySID starting at Edit Unit identified by Index Start Position. Identical Index Table Segments may be repeated in Partitions where required by the application.

Note: Further information on indexing CBE Essence is given in Section 11.1.9.

2. Where Edit Unit Byte Count is zero, Duration shall be the correct, non-zero value for the index table segment.
3. All Index Table Segments in each Partition shall have the same IndexSID value.
4. All Index Table Segments in one Index Stream (i.e., with the same IndexSID) shall have the same values of BodySID, Slice Count and Delta Entry Array.
5. For clarification from the table above, the size of the DeltaEntry Array is  $8 + (NDE * \text{Sizeof}(\text{DeltaEntry}))$
6. For clarification from the table above, the size of the IndexEntry Array is  $8 + (NIE * \text{Sizeof}(\text{IndexEntry}))$
7. The Edit Rate value in an Index Table Segment shall be the same as the Edit Rate value in all the Tracks that are indexed by that Index Table Segment. All Index Table Segments with the same IndexSID value shall have the same Edit Rate value.
8. The first Index Entry in the Index Entry Array within an Index Table Segment, shall give information about the byte offsets for the temporal location given by the Start Position item within that Index Table Segment.
9. The PosTableCount shall define the total number of Essence Elements which have fractional Positions recorded according to Section 11.1.3.
10. The value of ExtStartOffset shall be added to the offsets derived from the Index Table (whether CBE or VBE) to locate Essence data in an external file.
11. The value of ExtStartOffset shall not be used when locating Essence Container data within an MXF file.
12. If present, the value of VBEBByteCount shall equal the count of bytes of the last essence element in the last Edit Unit indexed by the Index Table Segment. This definition shall apply when indexing an internal Essence Stream or an external Essence file containing VBE Essence.
13. The value of VBEBByteCount shall not be used for CBE Essence. In this case, the total byte count is calculated as  $\text{EditUnitByteCount} \times \text{IndexDuration}$ .
14. The use of Index Tables for indexing external Essence is described in Section 11.5 below.
15.  $\text{Index Start Position} = N + \text{Origin}$ , where N is the number of the indexed Edit Unit.
16. Single Index Location and Forward Index Direction shall be omitted when the Index Table is either sparse or partial.
17. If the encoder cannot guarantee the validity of the Single Index Location, Single Essence Location and/or Forward Index Direction, the properties shall be omitted.

**Delta Entry Array** is a value where the order of the entries is significant and defines byte offset values along an incrementing timeline. The first 4 bytes shall be the number of Deltas Entries (NDE), the second 4 bytes shall be the length of each entry. The structure of each delta entry shall be as defined in Table 27.

Table 27 – Structure of Delta Entry Array

N	Item Name	Type	Len	Item UL	Req ?	Meaning	Default
1	NDE	UInt32	4	N/A	Req	Number of delta entries	
1	Length	UInt32	4	N/A	Req	Length of each delta entry	
Delta Entry	PosTableIndex	Int8	1	06.0E.2B.34 01.01.01.04 04.04.04.01 04.00.00.00	Req	-1=Apply Temporal Reordering 0= No temporal Reordering N=Index into PosTable [Note: SMPTE RP 210 definition value identifying that the element indexed is subject to temporal reordering or offsetting of edit units]	0
	Slice	UInt8	1	06.0E.2B.34 01.01.01.04 04.04.04.01 02.00.00.00	Req	Slice number in IndexEntry [Note: SMPTE RP 210 definition The number of the indexed section in the edit unit]	0
	Element Delta	UInt32	4	06.0E.2B.34 01.01.01.04 04.04.04.01 03.00.00.00	Req	Delta from start of Slice to this Essence Element [Note: SMPTE RP 210 definition The number of bytes from the start of the section to this element.]	

PosTableIndex is used to discover if this Essence Element has been temporally reordered or not. If the value is negative then the Temporal Offset Property of the IndexEntryArray shall be used to determine the temporal difference between presentation order and storage order of the Indexed Essence Element. If the value is positive then this Indexed Essence Element shall have a fractional temporal offset stored in the PosTable within the IndexEntry Array. The value of PosTableIndex is the index into this Table (1 is the first entry, 2 is the second etc.) If the value is zero, there shall be no reordering and no temporal offsetting for this Essence Element. For empty Essence Elements, the value of PosTableIndex shall be zero.

**Index Entry Array** is a value where the order of the entries is significant and defines values along an incrementing timeline. The first 4 bytes shall be the *Number of Index Entries* (NIE), the second 4 bytes shall be the length of each entry, which in turn shall depend on the *Number of Slices* (NSL) in this Index Table. The value of NSL shall be constant within all Segments of an Index Table. The structure of each Index Entry shall be as defined in Table 28.

Table 28 – Structure of Index Entry Array

N	Item Name	Type	Len	Item UL	Req ?	Meaning	Default
1	NIE	UInt32	4	N/A	Req	Number of index entries	
1	Length	UInt32	4	N/A	Req	Length of each Index Array entry	
Index Entry	Temporal Offset	Int8	1	06.0E.2B.34 01.01.01.04 04.04.04.02 03.00.00.00	Req	Offset in Edit Units from Display Order to Coded Order  [Note: SMPTE RP 210 definition The number of edit units by which this edit unit has been moved in the bitstream for the purpose of temporal reordering (e.g. MPEG)]	0
	Key-Frame Offset	Int8	1	06.0E.2B.34 01.01.01.04 04.04.04.02 04.00.00.00	Req	Offset in Edit Units to previous Key-Frame. The value is zero if this is a Key-Frame.  [Note: SMPTE RP 210 definition The offset in edit units from this edit unit to the previous Key-Frame edit unit (e.g. previous I-frame in MPEG-2)]	0
	Flags	EditUnitFlag	1	06.0E.2B.34 01.01.01.04 04.04.04.02 02.00.00.00	Req	Flags for this Edit Unit Bit 7: Random Access Bit 6: Sequence Header Bit 5: Forward prediction flag Bit 4: Backward prediction flag 00: I frame (no prediction) 10: P frame (forward prediction from previous frame) 01: B frame (backward prediction from future frame) 11: B frame (forward and backward prediction) Bits 0-3: reserved for use in SMPTE Essence mapping specifications.  [Note: SMPTE RP 210 definition Flags to indicate coding of elements in this edit unit]	80h
	Stream Offset	UInt64	8	06.0E.2B.34 01.01.01.04 04.04.04.02 01.00.00.00	Req	Offset in bytes from the start of the Essence Container of first Essence Element in this Edit Unit of stored Essence within the Essence Container Stream. See 11.1.4.  [Note: SMPTE RP 210 definition The offset of the edit unit within the container stream relative to the start of that container stream]	
	SliceOffset	NSL x UInt32	4* NSL	06.0E.2B.34 01.01.01.04 04.04.04.01 05.00.00.00	Opt	The offset in bytes from the Stream Offset to the start of this Slice.  [Note: SMPTE RP 210 definition List of the offsets within the edit unit of each indexed section (except the first)]	
	PosTable	NPE *Rational	8* NPE	06.0E.2B.34 01.01.01.05 04.04.04.01 08.00.00.00	Opt	The fractional position offset from the start of the Edit Unit to the synchronized sample in the indexed Essence Element (Figure 24).  [Note: SMPTE RP 210 definition List of the fractional temporal offsets of indexed elements relative to the indexed position]	

Conditions:

- i. The Temporal Offset shall be only applied for Essence Elements whose PosTableIndex value in the Element Delta is -1.
- ii. The Temporal Offset shall be zero (0) if the Essence Elements are not reordered.



- iii. The Key-Frame Offset shall be only applied for predictive coding schemes where a Key-Frame must be decoded in order to decode the indexed frame. It is the offset index to find the Key-Frame required to decode the indexed frame.
- iv. The Random Access flag bit shall be 1 if this Edit Unit is a random access point in the stream.
- v. The Sequence Header flag bit shall be 1 if this Edit Unit of stored Essence includes an MPEG Sequence Header. It shall be FALSE if the Edit Unit of stored Essence does not include a sequence Header or if the compression scheme does not make use of such a construct.
- vi. The forward and backwards prediction flags shall be set to zero if the compression scheme of the indexed Essence does not define inter-frame prediction.
- vii. The PosTable is an ordered list of signed fractional position offsets for indexed Essence Elements. If an Essence Element has an entry in this Array then the PosTableIndex Property in its DeltaEntry will be a positive Integer.
- viii. Figure 24 and its associated text shows the meaning of the sign of this Property.
- ix. The magnitude of the fractional offsets in the PosTable shall be bounded to such that it is equal or smaller than one (1).

### 11.3 Partial / Sparse Index Tables for VBE Essence

This specification allows an MXF Encoder to create several Index Table segments, each containing an array of one or more Index Entries. The most densely populated Index Table will have an Index Entry for every Edit Unit. This can be achieved by having a single large segment, or a number of contiguous segments where the first Index Entry in each segment is for the Edit Unit immediately following the last Index Entry in the preceding segment.

It is also permitted for there to be gaps between Index Table segments such that there are a number of Edit Units that are not indexed. Where there is an un-indexed gap between Index Table segments these tables are called Sparse Index Tables. Where all Index Table segments are contiguous, or there is only one segment, but not all Edit Units in the Essence Container are indexed, these tables are called Partial Index Tables.

A Sparse Index Table may include segments that index individual Edit Units spread through the Essence, or groups of Edit Units.

Both Sparse and Partial Index Tables provide less information than a complete table that indexes every Edit Unit. Random access to indexed locations within the Essence may be achieved as if the Index Table were complete, but un-indexed locations will require special handling by a decoder. Sparse or Partial Index Tables should only be used in applications where these limitations have been carefully considered.

Examples of Sparse Index Tables:

1. Where the Essence is divided into a number of Partitions, the first Edit Unit of stored Essence in each Partition is indexed. This allows the correct Partition to be located for any Edit Unit, but does not allow full random access.
2. The first Edit Unit in each GOP of a closed-GOP MPEG stream is indexed. This gives random access on a GOP-by-GOP basis.
3. One GOP of an MPEG stream is indexed for each 10 seconds of Essence. This gives course-grained random access to the Essence. This could be included as part of a specification that also adds extra constraints such as mandating that each of these indexed GOPs include a sequence header and that the GOP structure for each 10 second section be constant.

If a Sparse or Partial Index Table indexes Essence that is temporally reordered the Index Table shall not include any PosTableIndex value of -1, except all temporal reordering can be achieved with only the included entries.

#### 11.4 To Find the Byte Offset for an Essence Element (Informative)

To find a particular Essence Element within a given Edit Unit of stored Essence, first locate the Index Table Segment which contains the desired Edit Unit.

Apply the temporal offset to the Edit Unit number if appropriate, which will give you the required Index Entry for this Edit Unit. Inspecting the DeltaEntry Array for the desired Essence Element will provide the Slice Number and Element Delta (i.e. offset from the start of the Slice for the Essence Element). The byte offset of the Essence Element is given by adding:

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

#### 11.5 Using Index Tables for Internal Essence and External Essence

As described in Section 9.5.5, a Top-Level File Package may refer to internal Essence (i.e. Essence in the same MXF file), or to external Essence that may be stored in the Essence Container of another MXF file or in a file of a different file format.

The following sections define the semantics associated with zero or non-zero values of BodySID and IndexSID.

Note: The association of BodySID, IndexSID and Package ID values is defined by Essence Container Data sets. See Section 9.4.3.

##### 11.5.1 BodySID nonzero, IndexSID nonzero

An Index Table shall be present. It shall index the internal Essence Container data starting at the beginning of the Essence data in the MXF file.

##### 11.5.2 BodySID zero, IndexSID nonzero

An Index Table shall be present. The Index Table shall index the external Essence described by the Top-Level File Package that is associated to the Index Table.

The external Essence file may be an MXF file, or it may be a non-MXF file.

If the value of the Essence Container label of the File Descriptor does not equal the value that signals that the external file format is unknown (see Section 10.5.1), decoders can use its value to determine whether the external file is an MXF file or a non-MXF file. Otherwise, decoders shall make this determination by inspecting the external Essence file.

##### 11.5.2.1 Indexing an Essence Container in an external MXF File

The Properties in the Index Table shall have the same values as if the indexed Essence Container were internal, with the exception of BodySID, which shall be zero (0).

The indexed Essence Container in the external MXF file shall be described by a Top-Level File Package that has the same value of Package ID and identical values of Edit Rate and Track ID for all corresponding Essence Tracks.

##### 11.5.2.2 Indexing an external Non-MXF file

The format of the Essence byte stream in the external file shall be such that it could be the value of a clip-wrapped MXF Essence Container.

The MXF encoder shall index the Essence byte stream as if indexing the equivalent single Essence Element clip wrapped MXF Essence Container that holds the identical byte stream.

If the Edit Unit Byte Count Property of an Index Table Segment is non-zero and if the Essence byte stream in the external file begins at an offset greater than zero, the ExtStartOffset shall be present and its value shall equal the offset of the beginning of the Essence byte stream in the external file. The absolute byte offset of an Edit Unit of stored Essence within the external file shall be computed as the sum of the value of ExtStartOffset and the product of the Edit Unit Byte Count and the edit unit number.

If the Edit Unit Byte Count Property of an Index Table Segment is absent or zero and if the Essence byte stream in the external file begins at an offset greater than zero, the value of ExtStartOffset shall be set to equal the offset of the beginning of the Essence byte stream in the external file. The absolute byte offset of an Edit Unit of stored Essence within the external file shall be computed as the sum of the value of ExtStartOffset and the value of Stream Offset of the corresponding Index Array entry.

### 11.5.3 BodySID nonzero, IndexSID zero

An Index Table shall not be present for the internal Essence Container identified by BodySID.

### 11.5.4 BodySID zero, IndexSID zero

The Essence is external. There shall be no Index Table for this Essence

## 11.6 Additional Information (Informative)

### 11.6.1 Relationship between Top-Level File Package Essence Timeline Tracks and Index Entries

When every Edit Unit in an Essence Container is indexed and there is a Top-Level File Package which describes the indexed, interleaved Essence, the following statements are always true:

1. Each and every Essence Timeline Track of the Top-Level File Package has the same value of Edit Rate.
2. Each and every Essence Timeline Track of the Top-Level File Package has the same value of Origin.
3. The first Edit Unit of the Essence Container has an IndexStartPosition value of 0, including in case where Pre-Charge is used.
4. Origin equals the number of Edit Units before the first displayed Edit Unit of the Essence Container, i.e. the Edit Unit that corresponds to the Zero Point of the Package is indexed by IndexEntry[N], where N equals the number of Edit Units before the first displayed Edit Unit, and equals the Origin of the Tracks of the Top-Level File Package.
5. The Edit Unit at time  $(M / \text{Edit Rate})$  after the Zero Point of the Top-Level File Package is indexed by IndexEntry[M + N].
6. If there are no Edit Units in the Essence Container after time  $(K / \text{Edit Rate})$  after the Zero Point of the Top-Level File Package, then the Index Table has  $(K + N)$  Index Entries.

### 11.6.2 Look-up Algorithm for Conversion of Index Position to Stream Offset

The pseudo-code presented here is an informative aid to understand the algorithm which returns the Stream Offset of the T'th Edit Unit of a given sub-stream K. Variable names are the same as used in the text above. The example assumes that the Index Table is fully populated.

The code first locates the correct Index Table Segment by checking to see if the desired Edit Unit T is within the scope of the current Index Table Segment. Once that has been found, the conversion (if any) between coded order and display order is performed according to the example in Section 11.1.6. The Stream Offset for this Temporal Location can then be found. The substream (e.g. CBE sound) Slice is located in the Element Delta. If it is not the

first Slice then the offset to that Essence Element's Slice is added, and finally the Element Delta is added onto the previously calculated stream location to locate the start of this substream. Note that the offsets refer to the byte address within a stream, including all the Ks, Ls and Vs of each KLV item.

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

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

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

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

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

IndexTableSegment* I;      /* Current Index Table Segment */
Position T;                /* T is Index Position, i.e. the Edit Unit counted from start of the
                             Essence Container */
Substream K;               /* K is substream number, e.g. Sys==0,V==0,A==1,Dat==2 */

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

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

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

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

    UInt64 Result = I-> EditUnitIndex [T].StreamOffset;

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

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

    /* Now retrieve the offset to the start position - if any */
}
```

```

unit8 PosTableIndex = I->ElementDelta[K].PosIndex;
Rational pos = 0;
if (PosTableIndex>0)
    pos= I->EditUnitIndex [T].PosTable[PosTableIndex-1];

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

Return Result;
}

```

The Essence Container stream offset for substream K of the Edit Unit at time (N / Edit Rate) after the Zero Point of the Top-Level File Package is computed as

$$\text{StreamOffset}(N + \text{Origin}, K, I)$$

## 12 Random Index Pack

The Random Index Pack is a MXF component that can be used to find Partitions scattered throughout an MXF file. It shall be a Defined-Length Pack which defines the BodySID and byte offset to the start of each Partition (i.e. the first byte of the Partition Pack Key).

This Pack may be used by decoders to rapidly access Essence Containers or Index Tables and to find the Partitions to which an Index Table points.

The Random Index Pack is optional. If it exists, it shall follow the Footer Partition and shall be the last KLV item in the file.

MXF encoders should either place a Random Index Pack in the MXF file or set the correct, non-zero value of FooterPartiton in the Header Partition Pack.

### 12.1 Random Index Pack Key

The 16-byte SMPTE Universal Label of the Random Index Pack shall be as defined below.

**Table 29 – Random Index Pack Key Value**

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets and Packs
6	Registry Designator	05h	Defined-Length pack (no length fields)
7	Structure Designator	01h	Set/Pack Registry
8	Version Number	01h	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	File Structure sets and packs
14	Set/Pack kind	11h	Random Index Pack
15	Version	01h	Random Index Pack Version
16	Reserved	00h	

## 12.2 The Random Index Pack Value

The structure of the Random Index Pack value shall be as given below.

**Table 30 – Data Structure of the Random Index Pack Value**

N	Item Name	Type	Len	Item UL	Meaning	Default
	Random Index Metadata	Set Key	16	See Table 29		
↔	Length	BER Length	n		Overall Length of Pack	
N pairs	BodySID	UInt32	4	06.0E.2B.34 01.01.01.04 01.03.04.04 00.00.00.00	Stream ID of the Body in this Partition [Note: SMPTE RP 210 definition Essence (or its container) stream ID]	
	Byte Offset	UInt64	8	06.0E.2B.34 01.01.01.04 06.09.02.01 01.00.00.00	Byte offset from the first byte of the Header Partition Pack Key (which is numbered 0) to the 1 <sup>st</sup> byte of the Partition Pack Key [Note: SMPTE RP 210 definition Byte offset from start of file (byte 0) to 1st byte of Partition Pack key]	
	Length	UInt32	4	06.0E.2B.34 01.01.01.04 04.06.10.01 00.00.00.00	Overall Length of this Pack including the Set Key and BER Length fields [Note: SMPTE RP 210 definition Big-endian overall length of set or pack]	

If an MXF file contains 'N' Partitions (including Header Partition and Footer Partition) then the RIP Table shall contain 'N' pairs of values. Each pair of values relates the Byte Offset of the first byte of the Key of the Partition Pack of a Partition to the BodySID in that Partition. The pairs shall be stored in ascending Byte Offset order and every Partition shall be indexed if the Random Index Pack exists.

Note 1: When Partitions are relocated within a file (e.g. when moving Index Table Segments) the Random Index Pack must be recomputed or deleted.

Note 2: The last "Length" value is provided so that a decoder can inspect the last 4 bytes of an MXF file and use the value to skip immediately back to check for a Random Index Pack Key. Without this value, a decoder would have to parse from the file start or use another heuristic to detect the presence of a Random Index Pack.

## 12.3 Algorithm for using the Random Index Pack (Informative)

The following algorithm in pseudo-code allows the Random Index Pack to be found and read. Its use is application specific and dependent on the use of Index Tables.

```

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

```

## Annex A Specifications for Root Metadata Sets (normative)

This and later annexes define the MXF Structural Metadata Sets. Each Set is defined as part of a hierarchical inheritance model and Sets may define that other Properties are inherited from previously defined Sets. This is done to eliminate duplication of Properties that can lead to errors in the specification.

All MXF Structural Metadata Sets may be extended by adding Optional Properties. The addition of Required, Encoder Required, Decoder Required or Best Effort Properties shall result in the assignment of a new Group UL and thus the definition of a new Structural Metadata Set.

Note 1: A guide to the use of tables that define MXF sets is given in Section 4.4.

Note 2: SMPTE 377M-2004 defined some Sets in full (leading to many Properties being duplicated) and some Sets by adding Properties to already defined sets.

### A.1 Interchange Object

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Interchange Object	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Abstract Group that constitutes the Superclass of all MXF Sets and MXF Abstract Groups	
Instance UID	UUID	16	3C.0A	06.0E.2B.34 01.01.01.01 01.01.15.02 00.00.00.00	Req	Unique ID of this instance [Note: SMPTE RP 210 definition The ISO/IEC 11578 (Annex A) 16-byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	06.0E.2B.34 01.01.01.02 05.20.07.01 08.00.00.00	Opt	Generation Identifier [Note: SMPTE RP 210 definition Specifies the reference to an overall modification]	
Object Class	AUID	16	01.01	06.0E.2B.34 01.01.01.02 06.01.01.04 01.01.00.00	Opt	Class identifier of this Object. [Note: SMPTE RP 210 definition Specifies a reference to the definition of a class of object]	
ApplicationPlug-In Batch	Batch of StrongRef (Application Plug-In Object)	8+ 16n	dyn	06.0E.2B.34 01.01.01.0C 06.01.01.04 02.0e.00.00	Opt	Batch of Strong References to Application Plug-In Objects [Note: SMPTE RP 210 definition A batch of strong references to Application Plug-In Objects]	

This shall be an Abstract Class.

The Instance UID Property shall be used as the address of Strong References to instances of derived Classes.

If present, the Generation UID Property shall be used to weakly reference an Identification Set (via the value of its This Generation UID Property) that appears in the same Header Metadata instance.

If present, the Object Class Property shall identify the Subclass of InterchangeObject that is encoded in this Set. It shall only be used in accordance with other normative parts of the MXF specification (see Section 5.3).

If present, the ApplicationPlug-In Batch Property shall contain one or more Strong References to Application Plug-In Object instances (see Section 9.7). If there are no Application Plug-In Object instances that apply to the Interchange Object instance, the Property shall not be encoded.

## A.2 Preface

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Preface	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Preface Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Last Modified Date	Timestamp	8	3B.02	06.0E.2B.34 01.01.01.02 07.02.01.10 02.04.00.00	Req	Date and time of the last modification of the file [Note: SMPTE RP 210 definition Identifies date and time at the point of most recent modification of any item in the container]	
Version	Version Type	2	3B.05	06.0E.2B.34 01.01.01.02 03.01.02.01 05.00.00.00	Req	See 4.2. The value shall be 259 (i.e. v1.3) <u>Note 1:</u> For SMPTE 377M-2004 compliant files, the value of this Property is 258. <u>Note 2:</u> See 5.2.3 for provisions how to set the value when manipulating files that do not comply with this revision of SMPTE 377M-1.	
Object Model Version	UInt32	4	3B.07	06.0E.2B.34 01.01.01.02 03.01.02.01 04.00.00.00	Opt	Simple integer version number of Object Model The value, if present, shall be 1. [Note: SMPTE RP 210 definition Specifies the Internal Object Storage Mechanism Version Number]	
Primary Package	In-file WeakRef (Package)	16	3B.08	06.0E.2B.34 01.01.01.04 06.01.01.04 01.08.00.00	Opt	The primary Package in this file (see 8.5). The use of the primary Package may be defined by normative provisions in a specialized Operational Pattern. [Note: SMPTE RP 210 definition Specifies a reference to the primary Package in this file]	Instance UID value of the primary Material Package Set
Identifications	Array of StrongRef (Identification)	8+16n	3B.06	06.0E.2B.34 01.01.01.02 06.01.01.04 06.04.00.00	E/req	Array of strong references to Identification sets recording all modifications to the file [Note: SMPTE RP 210 definition Specifies a vector of references to modification identifiers]	



Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Content Storage	StrongRef (ContentStorage)	16	3B.03	06.0E.2B.34 01.01.01.02 06.01.01.04 02.01.00.00	Req	Strong reference to Content Storage Object [Note: SMPTE RP 210 definition Specifies a reference to the Packages and Essence Container Data Sets in a file]	
Operational Pattern	UL	16	3B.09	06.0E.2B.34 01.01.01.05 01.02.02.03 00.00.00.00	Req	Universal Label of the Operational Pattern which this file complies to (Partition Pack copies this value). Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Specifies the SMPTE Universal Label that locates an Operational Pattern]	
EssenceContainers	Batch of UL (Essence Containers)	8+ 16n	3B.0A	06.0E.2B.34 01.01.01.05 01.02.02.10 02.01.00.00	Req	A Batch of Universal Labels of Essence Containers used in or referenced by this file. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Batch of universal labels of all essence containers in the file]	
DM Schemes	Batch of UL (DM Schemes)	8+ 16n	3B.0B	06.0E.2B.34 01.01.01.05 01.02.02.10 02.02.00.00	Req	A Batch of Universal Labels of all the Descriptive Metadata Schemes used in this file. Individual UL values are listed in the Registry defined by SMPTE 400M (RP 224) [Note: SMPTE RP 210 definition An unordered Batch of Universal Labels of all the Descriptive Metadata schemes used in this file]	
Application Schemes Batch	Batch of UL (APScheme s)	8+ 16n	dyn	06.0E.2B.34 01.01.01.0C 01.02.02.10 02.03.00.00	Opt	A Batch of Universal Labels of all the Application Metadata Schemes used in this file. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition A batch of Universal Labels of all the Application Metadata schemes used in a file.]	
Is RIP Present	Boolean	1	dyn	06.0E.2B.34 01.01.01.0E 04.04.05.03 00.00.00.00	Opt	If a RIP exists in the file, the Boolean value shall be TRUE. Else (a RIP doesn't exist in the file), the Boolean value shall be FALSE. [Note: SMPTE RP210 definition: Specifies whether the file includes a Random Index Pack.]	

EssenceContainers is a Batch of ULs that identifies the types of Essence Containers used in this MXF file. This Batch shall contain all values that appear in the Essence Container Property of all File Descriptors (including Multiple Descriptors) of all Top-Level File Packages in the File.

Note: This definition of the contents of EssenceContainers in the Preface Set differs from the definition of the contents of EssenceContainers in the Partition Pack in Section 7.1.

The contents of EssenceContainers should be complete where possible.

If the encoder cannot guarantee the validity of “is RIP present”, the property shall be omitted.

If the Partition in which this Set is located is Closed, the values shall be complete and correct.

### A.3 Identification

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Identification	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Identification Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
This Generation UID	UUID	16	3C.09	06.0E.2B.34 01.01.01.02 05.20.07.01 01.00.00.00	Req	An immutable Generation Identifier to be referenced by other Sets (see section 7.5.2 for use of Generation UIDs) [Note: SMPTE RP 210 definition Specifies the reference to a particular modification.]	
Company Name	UTF-16 string	var	3C.01	06.0E.2B.34 01.01.01.02 05.20.07.01 02.01.00.00	Req	Manufacturer of the equipment or application that created or modified the file [Note: SMPTE RP 210 definition Specifies the name of the application provider]	
Product Name	UTF-16 string	var	3C.02	06.0E.2B.34 01.01.01.02 05.20.07.01 03.01.00.00	Req	Name of the application which created or modified this file [Note: SMPTE RP 210 definition Specifies the name of the application product]	
Product Version	Product Version	10	3C.03	06.0E.2B.34 01.01.01.02 05.20.07.01 04.00.00.00	Opt	Major, minor, tertiary, patch and release version number of this application (see 4.3) [Note: SMPTE RP 210 definition Specifies version information for the application]	
Version String	UTF-16 string	var	3C.04	06.0E.2B.34 01.01.01.02 05.20.07.01 05.01.00.00	Req	Human readable name of this application version [Note: SMPTE RP 210 definition Specifies version information for the application in textual form]	
Product UID	UUID	16	3C.05	06.0E.2B.34 01.01.01.02 05.20.07.01 07.00.00.00	Req	A unique identification for the product which created this file (defined by the manufacturer) [Note: SMPTE RP 210 definition Specifies a reference to the application product definition]	

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Modification Date	Timestamp	8	3C.06	06.0E.2B.34 01.01.01.02 07.02.01.10 02.03.00.00	Req	UTC time and date at which an application created or modified this file and created this Identification Set (see 4.3) <b>[Note: SMPTE RP 210 definition</b> Identifies date and time at the point of modification]	
ToolkitVersion	Product Version	10	3C.07	06.0E.2B.34 01.01.01.02 05.20.07.01 0A.00.00.00	Opt	Major, minor, tertiary, patch and release version of software or hardware codec used (see 4.3 ) <b>[Note: SMPTE RP 210 definition</b> Specifies the SDK version number for a modification]	
Platform	UTF-16 string	var	3C.08	06.0E.2B.34 01.01.01.02 05.20.07.01 06.01.00.00	Opt	Human readable name of the toolkit and operating system used. Best practice is to use the form "SDK name (OS name)" <b>[Note: SMPTE RP 210 definition</b> Specifies the platform on which the application was run]	

The optional Generation UID Property of the Interchange Object Class shall not be encoded in Identification Set instances.

Note: The rules for the addition and referencing of Identification Sets are defined in Section 7.5.2.

#### A.4 Content Storage

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Content Storage	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Content Storage Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
☐	Packages	Batch of StrongRef (Packages)	8+ 16n	19.01	06.0E.2B.34 01.01.01.02 06.01.01.04 05.01.00.00	Req	Batch of strong references to all Packages used in this file <b>[Note: SMPTE RP 210 definition</b> Specifies a unordered set of references to Packages]	
☐	EssenceContainer Data	Batch of StrongRef (Container Data)	8+ 16n	19.02	06.0E.2B.34 01.01.01.02 06.01.01.04 05.02.00.00	Opt	Batch of strong references to Essence Container Data sets used in this file <b>[Note: SMPTE RP 210 definition</b> Specifies a unordered set of references to Essence Data]	

## A.5 Essence Container Data

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Essence Container Data	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Essence Container Data Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Linked Package UID	PackageRef	32	27.01	06.0E.2B.34 01.01.01.02 06.01.01.06 01.00.00.00	Req	Identifier of the Package to which this Set is linked [Note: SMPTE RP 210 definition Specifies a reference to a Package associated with Essence Container Data]	
IndexSID	UInt32	4	3F.06	06.0E.2B.34 01.01.01.04 01.03.04.05 00.00.00.00	Opt	ID of the Index Table for the Essence Container to which this Set is linked [Note: SMPTE RP 210 definition Index table stream ID]	
BodySID	UInt32	4	3F.07	06.0E.2B.34 01.01.01.04 01.03.04.04 00.00.00.00	Req	ID of the Essence Container to which this Set is linked. The value of (0) indicates the Essence Container is external to the file. [Note: SMPTE RP 210 definition Essence (or its container) stream ID]	
Preceding Index Table	Boolean	1	dyn	06.0E.2B.34 01.01.01.0E 04.04.05.04 00.00.00.00	Opt	If a Complete Index Table for this Essence Container precedes all Essence that it indexes, the Boolean value shall be TRUE. Else, FALSE. [RP210 Specifies whether a Complete Index Table is in a partition preceding all Essence that it indexes.]	
Singular Partition Usage	Boolean	1	dyn	06.0E.2B.34 01.01.01.0E 04.06.02.07 00.00.00.00	Opt	See Section A.5.1 below. [RP210 Specifies whether the Partition is Singular.]	
Following Index Table	Boolean	1	dyn	06.0E.2B.34 01.01.01.0E 04.04.05.05 00.00.00.00	Opt	If a Complete Index Table for this Essence Container follows all Essence that it indexes, then, the Boolean value shall be TRUE. Else, FALSE. [RP210 Specifies whether a Complete Index Table is in a partition following all Essence that it indexes.]	
Is Sparse	Boolean	1	dyn	06.0E.2B.34 01.01.01.0E 04.04.05.06 00.00.00.00	Opt	If a Sparse Index Table is in the essence container, the Boolean value shall be TRUE. Else (a sparse Index Table isn't in the essence container) FALSE. [RP210 Specifies whether a sparse Index Table is in the essence container.]	

### A.5.1 Singular Partition Usage Semantics

If every Partition contains only an Essence Container Segment, or contains only one or more Index Table Segments, or neither of these segments, then the Singular Partition Usage property may be encoded, in which case its value shall be true.

If every Partition that contains an Essence Container Segment also contains one or more Index Table Segments, and the Footer does not contain any Index Table Segments, then the Singular Partition Usage property may be encoded, in which case its value shall be false.

If neither of the two conditions above are true or they are unknown, then the Singular Partition Usage property shall not be encoded.

Note: When Index Table Segments and Essence Container Segments are distributed among partitions, neither Preceding Index Table nor Following Index Table properties can express whether distributed Index Table Segments precede or follow the Essence they index. Preceding Index Table, Singular Partition Usage and Following Index Table properties can be used for example as follows.

Note: Each Partition in an MXF file can only contain Index Table Segments for a single Essence Container. Where a file contains more than one Essence Container it is still useful to be able to locate a Complete Index Table for each Essence Container at the start or end of the file. Encoders can write a sequence of partitions at the start of a file, each containing a Complete Index Table for an Essence Container. The first will be the Header Partition and must also contain Header Metadata (but no essence). The following partitions will be Body Partitions with no Header Metadata and no essence, just a Complete Index Table for the next Essence Container. Similarly, the last partitions in a file can contain a complete set of Index Tables for all Essence Containers in the file, the last will be the Footer Partition; the other Index Table Segments will be in Body Partitions (again with no essence and no Header Metadata)

## Annex B Specifications for the Generic Package (normative)


### B.1 Generic Package

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Generic Package	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Abstract Generic Package Group	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Package UID	PackageID	32	44.01	06.0E.2B.34 01.01.01.01 01.01.15.10 00.00.00.00	Req	The immutable Unique Package Identifier as a UMID [Note: SMPTE RP 210 definition Identifies the Metadata Object with a unique identifier]	
Name	UTF-16 string	var	44.02	06.0E.2B.34 01.01.01.01 01.03.03.02 01.00.00.00	Opt	Human readable Package name [Note: SMPTE RP 210 definition Identifies the AAF metadata object by name]	
Package Creation Date	Timestamp	8	44.05	06.0E.2B.34 01.01.01.02 07.02.01.10 01.03.00.00	Req	UTC date and time of creation of this Package (see 4.3) [Note: SMPTE RP 210 definition Identifies date and time at the point of creation.]	
Package Modified Date	Timestamp	8	44.04	06.0E.2B.34 01.01.01.02 07.02.01.10 02.05.00.00	Req	UTC date and time of last modification of this Package (see 4.3) [Note: SMPTE RP 210 definition Identifies date and time at the point of most recent modification of the Package]	
Tracks	Array of StrongRef (Tracks)	8+ 16n	44.03	06.0E.2B.34 01.01.01.02 06.01.01.04 06.05.00.00	Req	Array of Strong References to Tracks [Note: SMPTE RP 210 definition Specifies a vector of references to tracks]	

This shall be an Abstract Class.

### B.2 Generic Descriptor

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Generic Descriptor	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Abstract Generic Descriptor Group	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1		See A.1	See A.1	See A.1	See A.1	
Locators	Array of StrongRef (Locators)	8+ 16n	2F.01	06.0E.2B.34 01.01.01.02 06.01.01.04 06.03.00.00	Opt	Array of strong references to Locator Sets If present, Essence may be located external to the file. If there is more than one Locator Set an MXF decoder shall use them in the order specified. [Note: SMPTE RP 210 definition Specifies a vector of references to essence locators]	

 SubDescriptors	Array of StrongRef (SubDescriptors)	8+1 6n	dyn	06.0E.2B.34 01.01.01.09 06.01.01.04 06.10.00.00	Opt	Array of strong references to SubDescriptor Sets (see 10.5.4) [Note: SMPTE RP 210 definition Specifies a vector of an ordered set of references to SubDescriptor sets]	
--------------------------------------------------------------------------------------------------	-------------------------------------	-----------	-----	----------------------------------------------------------	-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--


This shall be an Abstract Class. It shall be the Superclass of all Descriptors.

A File Descriptor shall be a subclass of Generic Descriptor (see Annex F.2).

The Physical Descriptor shall be a subclass of Generic Descriptor.



Note: Specific Physical Descriptors are defined in other SMPTE Engineering Documents.

### B.3 SubDescriptor

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 SubDescriptor	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Superclass of all SubDescriptor Classes	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1		See A.1	See A.1	See A.1	See A.1	

This shall be an Abstract Class. It shall be the Superclass of all concrete SubDescriptors.

### B.4 Network Locator

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 Network Locator	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Network Locator Set for location with a URL	
 Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
URL String	UTF-16 string	var	40.01	06.0E.2B.34 01.01.01.01 01.02.01.01 01.00.00.00	Req	An URL indicating where the Essence may be found. [Note: SMPTE RP 210 definition Unique Resource Locator String]	

The URL String Property value of a Network Locator may be a URL or a URI. MXF Decoders shall provide support for the schemes defined in Sections B.4.1 through B.4.3. Recommended behavior for other schemes and invalid URLs / URIs is specified in Section B.4.4.

#### B.4.1 URL file://

A URL starting with file:// shall be treated as a fully qualified file path in accordance with RFC 3986.

An MXF application that supports external Essence shall support the file:// protocol.

Note: In the contents of the URL String value, reserved characters are escaped, e.g. ‘ ‘ is represented as ‘%20’.

### B.4.2 URL ftp://

A URL starting with ftp:// shall be treated as an external file available via the ftp protocol in accordance with RFC 3986.

A MXF application that supports external Essence, may support the ftp:// protocol.

### B.4.3 URIs

A string which does not match any known URL protocol shall be treated as a file path relative to the storage location of the MXF file containing the Network Locator.

Any MXF application supporting external Essence shall support relative URIs. All URIs shall be correctly formed according to RFC 3986.

### B.4.4 Handling invalid or unknown URLs and URIs

If a Network Locator string begins with "file://" (case insensitive) then it shall be a file scheme URL.

If the Network Locator string does not begin with "file://" (case insensitive), MXF decoders should test for the other URL schemes in RFC 3986 and issue a "not implemented" error, if appropriate.

If the Network Locator string does not match a URL scheme defined in RFC 3986, MXF decoders should treat it as a plain file path with no translation (i.e. don't convert %5C to \).

## B.5 Text Locator

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Text Locator	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Text Locator Set for location with a human-readable text string	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Locator Name	UTF-16 string	var	41.01	06.0E.2B.34 01.01.01.02 01.04.01.02 01.00.00.00	Req	Value of a human-readable locator text string for manual location of Essence <b>[Note: SMPTE RP 210 definition</b> A description of the physical location of media - e.g. which archive, place, rack, shelf, position on shelf]	

## B.6 Generic Track

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Track	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Abstract Generic Track Group	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	



Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Track ID	UInt32	4	48.01	06.0E.2B.34 01.01.01.02 01.07.01.01 00.00.00.00	D/req	ID of the Track in this Package (for linking to a SourceTrackID in a Source Clip)  [Note: According to B.7, this Property shall always be present and non-zero. [Note: SMPTE RP 210 definition Specifies the particular track within a Package]	
Track Number	UInt32	4	48.04	06.0E.2B.34 01.01.01.02 01.04.01.03 00.00.00.00	Req	Number used link the Track to an Essence Element in the Essence Container.  [Note: SMPTE RP 210 definition Specifies a reference to the codec used to create Essence Data]	0
Track Name	UTF-16 string	var	48.02	06.0E.2B.34 01.01.01.02 01.07.01.02 01.00.00.00	Opt	Human-readable name of the Track type.  [Note: SMPTE RP 210 definition Specifies the particular track within a package by a name]	
Sequence	StrongRef (Sequence)	16	48.03	06.0E.2B.34 01.01.01.02 06.01.01.04 02.04.00.00	Req	Strong reference to the Sequence Set.  [Note: SMPTE RP 210 definition Specifies a reference to a segment of a track]	

This shall be an Abstract Class.

The value of the Track Number should be set to zero in all Material Package and Lower-Level Source Package Essence Tracks and in all Descriptive Metadata Tracks.

Note: Some MXF encoders create files that contain non-zero Track Number Properties in Material Package Essence Tracks.

Non-zero values of the Track Number Property in Essence Tracks of Material or Lower-Level Source Packages, and non-zero values of the Track Number Property in Descriptive Metadata Tracks should be treated as Dark Metadata.

## B.7 Track ID Usage


Annex B.6 defines the Track ID Property as D/Req (decoder required). However, the Track ID shall be the unique identifier of a Track Object within a Package. It shall always be encoded such that all Tracks within Packages can be referenced in all cases (as required to support Package derivation chain references).

In SMPTE 377M-2004, MXF encoders were not required to encode the Track ID Property, which means there is a potential legacy of MXF files in Tracks might not have a Track ID. MXF decoders shall use the value of zero (0) as Track ID for Tracks where the Track ID Property is missing.

A Track ID value of zero (0) is deprecated, and shall not be used by MXF encoders conforming to this version of the MXF file format specification.

Note: This constraint is in order to avoid confusion with the Distinguished Values used in a Source Clip Object when denoting the end of the Package derivation chain (which is a Source Clip Object with zero valued SourcePackageID, SourceTrackID and StartPosition Properties).

## B.8 Structural Component




Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 Structural Component	Group UL	16		As defined in 9.6 (see Table 19)	Req	Defines the Abstract Structural Component Group	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Data Definition	UL	16	02.01	06.0E.2B.34 01.01.01.02 04.07.01.00 00.00.00.00	Req	Specifies the data type of this Set. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Specifies the basic Essence type of a component]	
Duration	Length	8	02.02	06.0E.2B.34 01.01.01.02 07.02.02.01 01.03.00.00	Opt	Duration of Sequence (in units of Edit Rate) [Note: SMPTE RP 210 definition The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	

This shall be an Abstract Class.

If the Structural Component is strongly referenced from a Timeline Track, Duration shall be treated as a B.Effort Property with a Distinguished Value = -1.

If the Structural Component is strongly referenced from a Static Track, the Duration Property shall be omitted.

## B.9 Sequence

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
 Length	BER Length	var			Req	Set length (see 9.3)	
All items in B.8 except the Key or Group UL and Length, if present	See B.8	See B.8	See B.8	See B.8	See B.8	See B.8	
 Structural Components	StrongRefArray (Structural Component)	8+ 16n	10.01	06.0E.2B.34 01.01.01.02 06.01.01.04 06.09.00.00	Req	Array of strong references to instances of specific subclasses of Structural Component [Note: SMPTE RP 210 definition Specifies a vector of references to the clips and transitions in the sequence]	

The following limitations shall apply to the contents of the Structural Components Property:

1. If the Sequence is strongly referenced from an Essence Track, the Structural Components Property shall strongly reference Source Clips or Fillers.
2. If the Sequence is strongly referenced from a Timecode Track, the Structural Components Property shall strongly reference Timecode Components.
3. If the Sequence is strongly referenced from a Descriptive Metadata Track, the Structural Components Property shall strongly reference DM Segments or DM Source Clips.

If the Sequence is strongly referenced from a Timeline Track, the value of Duration (see Annex B.8) shall equal the sum of the values of Duration of all Structural Components.

If the Sequence is strongly referenced from an Event Track, the value of Duration (see Annex B.8) shall equal the difference between the maximum value of Event Start Position +Duration for all Structural Components and the minimum value of Event Start Position +Duration for all Structural Components.

### B.10 Source Clip

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Source Clip	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Source Clip Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.8 except the Key or Group UL and Length, if present	See B.8	See B.8	See B.8	See B.8	See B.8	See B.8	
	Start Position	Position	8	12.01	06.0E.2B.34 01.01.01.02 07.02.01.03 01.04.00.00	Req	Offset into the Track of the referenced Package measured in Edit Units of the Track containing this Source Clip relative to the Zero Point of the referenced Package <b>[Note: SMPTE RP 210 definition</b> The relative start time from an origin within the track of a clip, expressed in edit units]	
	SourcePackageID	PackageRef	32	11.01	06.0E.2B.34 01.01.01.02 06.01.01.03 01.00.00.00	Req	ID of referenced Package as a UMID. The value shall be 32 zero valued bytes to terminate the source reference chain. <b>[Note: SMPTE RP 210 definition</b> Specifies the reference to a precursor]	
	SourceTrackID	UInt32 (Track ID)	4	11.02	06.0E.2B.34 01.01.01.02 06.01.01.03 02.00.00.00	Req	Track ID value of the referenced Track within the referenced Package. The value shall be zero (0) to terminate the source reference chain. <b>[Note: SMPTE RP 210 definition</b> Specifies the track within the referenced precursor]	

Start Position shall be present if the Source Clip references (via the values of SourcePackageID and SourceTrackID) a Timeline Track or an Event Track.

Start Position shall be omitted if the Source Clip references a Static Track.

### B.11 Filler

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Filler	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Filler Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.8 except the Key or Group UL and Length, if present	See B.8	See B.8	See B.8	See B.8	See B.8	See B.8	

The Filler shall only be referenced from the Sequence of a Timeline Track.

The Filler shall be referenced from the Sequence of a Timeline Track if there is no defined Essence or Descriptive Metadata for the value of Duration.

The Duration Property of the Filler shall be greater than 0.

If a Filler Object is played on a Sound Essence Track, the output shall be silence.

If a Filler Object is played on a Picture Essence Track, the output should be black.

If a Filler Object is played on a Data Essence Track, applications shall choose any appropriate blank Essence to output.

Note 1: The Filler serves an entirely different function as the KLV Fill Item (see Section 6.3.3). The KLV Fill item is used to add bytes in the serialized bitstream. The Filler is used to signal empty Essence along the timeline of a Track.

Note 2: It is not guaranteed that the Filler is supported by SMPTE 377M-2004 MXF decoders.

## B.12 Timeline Track

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timeline Track Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.6 except the Key or Group UL and Length, if present	See B.6	See B.6	See B.6	See B.6	See B.6	See B.6	
	Edit Rate	Rational	8	4B.01	06.0E.2B.34 01.01.01.02 05.30.04.05 00.00.00.00	Req	Edit Rate of Track [Note: SMPTE RP 210 definition Specifies the timeline rate in hertz]	
	Origin	Position	8	4B.02	06.0E.2B.34 01.01.01.02 07.02.01.03 01.03.00.00	Req	An Offset used to resolve timeline references to this Track. The start of the Track has this timeline value measured in Edit Units. [Note: SMPTE RP 210 definition Specifies the point, in edit units, in a track from which relative times are measured.]	

## B.13 Track Event

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Event Track	Set Key	16		As defined in 9.6 (See Table 17)	Req	Defines the Event Track Set	
↔	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.6 except the Key or Group UL and Length, if present	See B.6	See B.6	See B.6	See B.6	See B.6	See B.6	

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Event Edit Rate	Rational	8	49.01	06.0E.2B.34 01.01.01.02 05.30.04.02 00.00.00.00	Req	Edit Rate of Track [Note: SMPTE RP 210 definition Specifies the timeline rate in hertz]	
Event Origin	Position	8	49.02	06.0E.2B.34 01.01.01.05 07.02.01.03 01.0B.00.00	Opt	An Offset used to resolve timeline references to this Track. The start of the Track has this timeline value measured in Edit Units. [Note: SMPTE RP 210 definition Specifies the point, in edit units, in an event track from which relative times are measured.]	0

#### B.14 Static Track

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Static Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Static Track Set	
↔	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.6 except the Key and Length	See B.6	See B.6	See B.6	See B.6	See B.6	See B.6	

#### B.15 Timeline Track (Timecode)

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timeline Track Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.12 except the Key or Group UL and Length, if present	See B.12	See B.12	See B.12	See B.12	See B.12	See B.12	

If a Package has exactly one Timecode Track, then this track defines the default timecode. If a Package has more than one Timecode Track, the Track Number Property of exactly one Timecode Track shall be set to 1 to indicate the default time code when playing the Package. For the remaining Timecode Tracks, the Track Number Property, if present, shall be set to zero. All other values for Timecode Track Numbers are SMPTE reserved.

For the use and semantics of Timecode Tracks see Section 9.4.

## B.16 Sequence (Timecode)

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in B.9 except the Key or Group UL and Length, if present	See B.9	See B.9	See B.9	See B.9	See B.9	See B.9	

## B.17 Timecode Component

This Set is used to define continuous timecode over the duration of this Component. This Set may be used in any Package.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Timecode Component	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timecode Component Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in B.8 except the Key or Group UL and Length, if present	See B.8	See B.8	See B.8	See B.8	See B.8	See B.8	
Start Timecode	Position	8	15.01	06.0E.2B.34 01.01.01.02 07.02.01.03 01.05.00.00	Req	Starting timecode (converted to integer frame count from 00:00:00:00) [Note: SMPTE RP 210 definition The timecode within a track at the starting point of the Essence]	
Rounded Timecode Base	UInt16	2	15.02	06.0E.2B.34 01.01.01.02 04.04.01.01 02.06.00.00	Req	Nearest Integer frames per second [Note: SMPTE RP 210 definition e.g. 24, 25, 30, 48, 60]	
Drop Frame	Boolean	1	15.03	06.0E.2B.34 01.01.01.01 04.04.01.01 05.00.00.00	Req	True = Drop frame timecode in use [Note: SMPTE RP 210 definition Specifies whether timecode is drop frame (Non-drop Frame = 0)]	

Note 1: An algorithm for determining the Rounded Timecode Base= integer\_part\_of (Edit Rate + 0.5).

Note 2: Timecode Components in the MXF header metadata represent piecewise linear synthetic timecode values. They contain an integer frame count of the first frame to which the timecode component relates, the Rounded Timecode Base (frames per second) and a Drop Frame flag. Based in this information only the values of hours, minutes, seconds and frames of a corresponding SMPTE ST 12-1 timecode can be computed. Timecode Components do not provide a mechanism to encode SMPTE ST 12-1 binary groups or SMPTE ST 12-1 binary group flags. As a consequence, the labels for SMPTE ST 12-1 timecode with active user bits or SMPTE ST 309 timecode are not applicable for use as labels for the Data Definition Properties of Timecode Components.

**B.18 Timeline Track (Picture)**

Note: The number of Picture Track sets is determined by the number of Picture Elements in the Essence Container. See Section 9.5.5 Top-Level File Packages.

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timeline Track Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.12 except the Key and Group UL and Length, if present	See B.12	See B.12	See B.12	See B.12	See B.12	See B.12	

**B.19 Sequence (Picture)**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.9 except the Key or Group UL and Length, if present	See B.9	See B.9	See B.9	See B.9	See B.9	See B.9	

**B.20 Source Clip (Picture)**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Source Clip	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Source Clip Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.10 except the Key or Group UL and Length, if present	See B.10	See B.10	See B.10	See B.10	See B.10	See B.10	

## B.21 Timeline Track (Sound)

Note: The number of Sound Track sets is determined by the number of Sound Elements present in the Essence Container. See Section 9.5.5 Top-Level File Packages.

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☰	Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timeline Track Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.12 except the Key or Group UL and Length, if present	See B.12	See B.12	See B.12	See B.12	See B.12	See B.12	

## B.22 Sequence (Sound)

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☰	Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.9 except the Key or Group UL and Length, if present	See B.9	See B.9	See B.9	See B.9	See B.9	See B.9	

## B.23 Source Clip (Sound)

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☰	Source Clip	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Source Clip Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.10 except the Key or Group UL and Length, if present	See B.10	See B.10	See B.10	See B.10	See B.10	See B.10	
	Channel IDs	Array of Uint32	8+ 4n	1103	06.0E.2B.34. 01.01.01.07. 06.01.01.03. 07.00.00.00	Opt	Ordered Array of Channel ID referenced by this Source Clip. [Note: SMPTE RP 210 definition Specifies an ordered array of channels within the referenced precursor]	
	Mono Source Track IDs	Array of Uint32	8+ 4n	1104	06.0E.2B.34. 01.01.01.08. 06.01.01.03. 1) 08.00.00.00	Opt	Ordered Array of Track IDs referenced by this Source Clip. [Note: SMPTE RP 210 definition Specifies an ordered array of tracks within the referenced precursor]	

The optional Channel IDs and Mono Source Track IDs properties may be used when a package references a lower-level package such that the reference:

- extracts one or two channels of audio from an N-channel lower-level package



- combines mono tracks from the lower-level package to form a multi-channel package

These properties are not intended to be used either for a generalized N-channel to M-channel mapping mechanism or as global identifiers. Their scope is limited to a single track within a top-level file package, as identified by the Source Track ID and Package ID properties in the case of a Source Clip (see Section B.10.)

A Source Clip shall not simultaneously contain both a Channel IDs property and a Mono Source Track IDs property

### B.23.1 Optional Mapping of Subchannels of Sound Elements to Material Package

If present, the Channel IDs property shall reference one or two channels out of N-channel essence described by a top-level file package. This is used to break out a single track of multi-channel sound into individual mono sound tracks, or to identify a stereo sound pair within a multi-channel essence container.

Each essence container specification that uses this mechanism shall define any specific semantics of Channel ID for that essence..

Note: In the case of sound compressed and mapped into AES pairs according to SMPTE ST 337, each AES pair can contain multiple SMPTE ST 337 streams and hence channels.

### B.23.2 Optional combination of Source Package Tracks

If present, the optional Mono Source Track IDs property shall identify the single channel tracks of lower-level source packages which were combined to form a multi-channel track.

When the Mono Source Track IDs property is present, the value of the Source Track ID property of the Source Clip (see Section B.10) shall be ignored.

Note: The Mono Source Track IDs property is used on a top-level file package to describe the transfer of several individual single-channel inputs from a source device into a multi-channel essence container.

## B.24 Timeline Track (Data)

Note: The number of Data Track sets is determined by the number of Data Elements present in the Essence Container. See Section 9.5.5 Top-Level File Packages.

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Timeline Track Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.12 except the Key or Group UL and Length, if present	See B.12	See B.12	See B.12	See B.12	See B.12	See B.12	

**B.25 Sequence (Data)**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in B.9 except the Key or Group UL and Length, if present	See B.9	See B.9	See B.9	See B.9	See B.9	See B.9	

**B.26 Source Clip (Data)**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Source Clip	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Source Clip Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in B.10 except the Key or Group UL and Length, if present	See B.10	See B.10	See B.10	See B.10	See B.10	See B.10	

**B.27 Dm Tracks**

Descriptive Metadata Tracks may be one of three types: a Timeline Track, an Event Track or a Static Track. A DM Timeline Track represents a contiguous sequence of Source Clips or Segments. A DM Timeline Track shall not have overlapping DM Source Clips or DM Segments. A DM Event Track represents a number of, possibly independent, Events occurring in time. The Events may occur simultaneously, may be instantaneous or may have a duration. A DM Event Track may have overlapping DM Source Clips or DM Segments and the DM Source Clips or DM Segments may be zero duration. A DM Static Track holds unchanging metadata. A DM Static Track shall represent a static event and DM Source Clips and DM Segments in a Static Track shall not have a Duration Property.

Note: DM Tracks are only present when there is Descriptive Metadata in the file. This includes DM Segments and DM Source Clips.

**B.27.1 Timeline Track (DM)**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Track Set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in B.12 except the Key or Group UL and Length, if present	See B.12	See B.12	See B.12	See B.12	See B.12	See B.12	

**B.27.2 Event Track (DM)**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Event Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Event Track Set	
↔	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.13 except the Key or Group UL and Length, if present	See B.13	See B.13	See B.13	See B.13	See B.13	See B.13	

**B.27.3 Static Track (DM)**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Static Track	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Static Track Set	
↔	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.14 except the Key or Group UL and Length, if present	See B.14	See B.14	See B.14	See B.14	See B.14	See B.14	

**B.28 Sequence (DM)**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
	Sequence	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sequence Set	
	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.9 except the Key or Group UL and the Length, if present	See B.9	See B.9	See B.9	See B.9	See B.9	See B.9	

**B.29 Segment**

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
	Segment	Set Key	16		As defined in 9.6 (see Table 18)	Req	Defines the Abstract Segment Group	
	Length	BER Length	var			Req	Set length (see 8.3)	
	All items in B.8 except the Key or Group UL and the Length, if present	See B.8	See B.8	See B.8	See B.8	See B.8	See B.8	

This shall be an Abstract Class.

**B.30 Event**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Event	Group UL	16		Defined by the Comment Market type. As defined in 9.6 (see Table 18)	Req	Defines the Abstract Event Group	
All items in B.29 except the Key or Group UL and the Length, if present	See B.29	See B.29	See B.29	See B.29	See B.29	See B.29	
Event Start Position	Position	8	06.01	06.0E.2B.34 01.01.01.02 07.02.01.03 03.03.00.00	Opt	Offset into the Descriptive Metadata Track in Edit Units [Note: SMPTE RP 210 definition Specifies the Starting Time of an Event in edit units, relative to the origin]	
Event Comment	UTF-16 String	Var	06.02	06.0E.2B.34 01.01.01.02 05.30.04.04 01.00.00.00	Opt	Description of the Descriptive Metadata Framework [Note: SMPTE RP 210 definition User-provided Comment Text for an event]	

This shall be an Abstract Class.

Event Start Position shall be required if the Comment Marker is strongly referenced from an Event Track.

Event Start Position shall be omitted if the Comment Marker is strongly referenced from a Timeline Track or from a Static Track.

**B.31 Comment Marker**

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Comment Marker	Group UL	16		As defined in 9.6 (see Table 18, Table 17)	Req	Defines the Abstract Comment Marker Group	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in B.30 except the Key or Group UL and the Length, if present	See B.30	See B.30	See B.30	See B.30	See B.30	See B.30	

This shall be an Abstract Class.

## B.32 DM Segment

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
DM Segment	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Descriptive Metadata Segment Set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in B.31 except the Key or Group UL and the Length, if present	See B.31	See B.31	See B.31	See B.31	See B.31	See B.31	
Track IDs (DM Segment)	Batch of UInt32 (Track ID)	8+ 4*n	61.02	06.0E.2B.34 01.01.01.04 01.07.01.05 00.00.00.00	D/req	Specifies an unordered list of Track ID values that identify the Tracks in this Package to which this DM Framework refers (if omitted, refers to all Essence Tracks)  [Note: SMPTE RP 210 definition Specifies an unordered list of track ID values that identify descriptive metadata tracks by containment]	
DM Framework	StrongRef (DM Framework)	16	61.01	06.0E.2B.34 01.01.01.05 06.01.01.04 02.0C.00.00	D/req	Strong Reference to the Descriptive Metadata Framework  [Note: SMPTE RP 210 definition Strong Reference to the Descriptive Metadata Framework]	
Descriptive Metadata Plug-In ID	UUID	16	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 0e.00.00.00	Opt	The immutable ID of this DM Plug-in instance  [Note: SMPTE RP 210 definition Universal Label of this DM Plug-in instance]	
Descriptive Metadata Scheme	UL	16	dyn	06.0E.2B.34 01.01.01.0C 04.06.08.04 00.00.00.00	Opt	The Universal Label of the Descriptive Metadata Scheme that is referenced by the DM Framework Property.  Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224).  [Note: SMPTE RP 210 definition The Universal Label of the Descriptive Metadata scheme that is referenced by the DM Framework property]	
Descriptive Metadata Application Environment ID.	UTF-16 string	var	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 10.00.00.00	Opt	The RFC 3986 Uniform Resource Identifier that identifies the application to which the information in this DM plug-in applies  [Note: SMPTE RP 210 definition The RFC 3986 Uniform Resource Identifier that identifies the application to which the information in this DM plug-in applies]	

Note: The Descriptive Metadata Segment is also a Subclass of the AAF Comment Marker Class. It has incorporated within it a DM Track IDs item which specifies the target Tracks of the Descriptive Metadata. It includes a Strong Reference to a DM Framework item to contain the Descriptive Metadata for this Segment.

## B.33 DM Source Clip

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
DM Source Clip	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Descriptive Metadata Source Clip Set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in B.10 except the Key or Group UL and the Length, if present	See B.10	See B.10	See B.10	See B.10	See B.10	See B.10	
Track IDs (DM Source Clip)	Batch of UInt32 (Track ID)	8+ 4*n	61.03	06.0E.2B.34 01.01.01.05 01.07.01.06 00.00.00.00	D/req	Specifies an unordered list of Track ID values that identify the target Tracks in this Package to which the referenced Descriptive Metadata refers (if omitted, refers to all Essence Tracks) <b>[Note: SMPTE RP 210 definition</b> Specifies an unordered list of track ID values that identify metadata source tracks by reference]	

Note 1: This is a Source Clip which can be Strongly Referenced by Sequence (Descriptive Metadata) in order to provide a linking / derivation mechanism between Descriptive Metadata in Material Package, File Package and Source Packages.

A DM Source Clip shall only be referenced by a Sequence in a (Timeline) Track.

Note 2: The DM Source Clip is used in the same way as a Source Clip (see Figure 11). The Start Position and Duration specify the portion of the Source Metadata which is relevant for this Source Clip. The Source Metadata can be a DM Segment in any of the Tracks listed in Annex B.17.

## B.34 Package Marker Object

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Package Marker Object	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Package Marker Object	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1	See A.1	See A.1	See A.1	See A.1	See A.1	
Timebase ReferenceTrack ID	UInt32 (Track ID)	4	dyn	06.0E.2B.34 01.01.01.0C 06.01.01.03 0E.00.00.00	Req	Specifies the value of the Track ID of the target Track in the Material Package that provides the Edit Rate. <b>[Note: SMPTE RP 210 definition</b> Specifies the value of the Track ID of the target Track in the Material Package that provides the Edit Rate.]	
Package Mark In Position	Position	8	dyn	06.0E.2B.34 01.01.01.0A 07.02.01.03 01.0E.00.00	D/R eq	Start of the optional sub-section on the Material Package Timebase Reference Track timeline. <b>[Note: SMPTE RP 210 definition</b> Specifies an optional start of playback position. The position is measured in edit units relative to the origin, and applies to all tracks of the material]	
Package Mark Out Position	Position	8	dyn	06.0E.2B.34 01.01.01.0A 07.02.01.03 02.04.00.00	D/R eq	Stop of the optional sub-section on the Material Package Timebase Reference Track timeline. <b>[Note: SMPTE RP 210 definition</b> Specifies an optional end of playback position. The position is measured in edit units relative to the origin, and applies to all tracks of the material]	

Package Mark In Position and Package Mark Out Position specify an optional sub-section of a Material Package, which may be played as an alternative to the full Material Package.

The value of Timebase Reference Track ID shall specify the Track ID value of the Essence Timeline Track in the Material Package that strongly references the Package Marker Object instance. Package Mark In Position and Package Mark Out Position shall be applied relative to the timeline defined by that Track.

An application that implements the optional Package Marker Object shall have the following behavior when playing a Material Package:

1. If the Material Package has a Package Marker Object and the Package Mark-In Position Property is present, the default play start position (i.e. the first Edit Unit to be played) shall be the position defined by the value of the Package Mark-In Position Property. Otherwise, the default play start position shall be at the Zero Point of the Package.
2. If the Material Package has a Package Marker Object and the Package Mark-Out Position Property is present, the default play stop position (i.e. the first Edit Unit that is not played) shall be the Edit Unit of the Essence Timeline Reference Track position defined by the value of the Package Mark-Out Position Property. Otherwise, the default stop position shall be at Package Duration.

Applications that implement the optional Package Marker Object may also be configured to ignore Package Marker Objects when playing Material Packages.

## Annex C Specification of the Application-Specific Metadata Plug-In Mechanism Sets (normative)

The use of these sets is defined in Section 9.7 and illustrated in Figure 18.

### C.1 Application Object

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Application Object	Group UL	16		As defined in 9.6 (See Table 17)	Req	Defines the Abstract Superclass of the Application Plug-in Objects and Application Referenced Objects	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1		See A.1	See A.1	See A.1	See A.1	
Base Class	AUID	16	dyn	06.0E.2B.34 01.01.01.0C 06.01.01.04 01.0B.00.00	Opt	Class Identifier of the immediate Superclass defined in an MXF specification that this Object extends.  [Note: SMPTE RP 210 definition Specifies a reference to the definition of a class of object]	

This shall be an Abstract Class.

### C.2 Application Plug-In Object

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Application Plug-In Object	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Application Plug-in Object Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in C.1 except the Key or Group UL and the Length, if present	See C.1		See A.1	See C.1	See C.1	See C.1	
Application Plug-In Instance ID	UUID	16	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 0d.00.00.00	Req	An immutable ID of this application metadata plug-in instance.  [Note: SMPTE RP 210 definition UUID of this application metadata plug-in instance.]	
Application Scheme	UL	16	dyn	06.0E.2B.34 01.01.01.0C 04.06.08.03 00.00.00.00	Req	Universal Label of the Application Metadata Scheme contained in this Plug-In Object  [Note: SMPTE RP 210 definition Contains the Universal Label of the Application Metadata scheme contained in this Plug-In Object.]	
Application Environment ID	UTF-16 string	var	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 0f.00.00.00	Opt	RFC 3986 Uniform Resource Identifier that identifies the application to which the information in this Plug-In Object applies.  [Note: SMPTE RP 210 definition The RFC 3986 Uniform Resource Identifier that identifies the application to which the information in this Plug-In Object applies.]	



Any extension Property defined by the Application Metadata Scheme	The type of the specific Property defined by the Application Metadata Scheme	var	Static, if the UL appears in Annex H, dyn otherwise	Appropriate value from SMPTE ST 335 (RP 210) or UUID defined according to the Application Metadata Scheme	As defined by the application	As defined by the Application Metadata Scheme	As defined by the Application Metadata Scheme
-------------------------------------------------------------------	------------------------------------------------------------------------------	-----	-----------------------------------------------------	-----------------------------------------------------------------------------------------------------------	-------------------------------	-----------------------------------------------	-----------------------------------------------

If the Application Plug-In does not Subclass the Set it extends (i.e. the Set that references the Application Metadata Plug-In Object), the Object Class value shall be identical to the Class ID (e.g. KLV Key) of the extended Object.

The last row of the table signifies that any number of Properties may be added to Application Plug-In Object instances, provided the following two rules are observed:

1. The UL or UUID values that identify the Property shall be unique within the Application Plug-In Object instance.
2. The UL values that identify the Property shall be different from the ULs of all Required, Best Effort, Decoder Required or Encoder Required Properties of the Class that the Application Plug-In Object instance extends.

Note 1: This means that the only Properties that cannot be added are the ones already defined to belong to the Application Plug-In Object Class as well as the ones which are required Properties of the Class that is extended by the Application Plug-In Object instance.

SMPTE ST 395 and SMPTE ST 377-1 permit the addition of optional Properties to registered MXF Header Metadata Sets such that the Universal Label that identifies the group in the register defined by SMPTE ST 395 remains the same. From the perspective of the Application-Specific Metadata plug-in mechanism, this means that a metadata element that is defined by an Application Metadata Scheme for use within an Application Plug-In Object could also appear as an optional Property of the MXF Header Metadata Set that the Plug-In Object extends. The behavior of implementations in this case that support the specific Application Metadata Scheme shall be defined by the Application Metadata Scheme specification.

Table 3 defines that MXF decoders may (or may not) decode optional Properties of MXF Header Metadata Sets. Therefore, the Application Scheme specification may define one of the following behaviors for each individual Property:

- i. To discard the value of the optional Property in the MXF Header Metadata Class and to use the value in the Application Plug-In Object,
- ii. To use the value of the optional Property in the MXF Header Metadata Class and to discard the value in the Application Plug-In Object,
- iii. To use the value of the optional Property in the MXF Header Metadata Class and to use the value in the Application Plug-In Object.

Option iii applies to the case where the semantics of both Properties are known, are different and are supported by the MXF application that implements the Application Metadata Scheme.

Note 2: Application Plug-In Objects are encoded as MXF Local Sets. This means that, within Application Plug-In Object instances, individual Properties are identified by the two byte Local Tag that is associated to the Property UL or UUID via the Primer Pack.

### C.3 Application Referenced Object

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Application Referenced Object	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Application Referenced Object Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in C.1 except the Key or Group UL and the Length, if present	See C.1	See C.1	See C.1	See C.1	See C.1	See C.1	
	Linked Application Plug-In Instance ID	Global WeakRef (Application Plug-In Object)	16	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 0b.00.00.00	Req	Global Weak Reference to the Application Plug-In Object that (directly or indirectly) strongly references this Application Metadata Referenced Object Set. Note: This is infile. <b>[Note: SMPTE RP 210 definition</b> Global Weak Reference to the Application Plug-In Object that (directly or indirectly) strongly references this Application Metadata Referenced Object Set.]	
	Any Property defined by the Application Scheme Specification for the Class that is contained in the Application Referenced Object instance	The type of the specific Property defined by the Application Metadata Scheme	var	Static, if the UL appears in Annex H, dyn otherwise	Appropriate value from SMPTE ST 335 (RP210) or UUID defined according to the Application Metadata Scheme	As defined by the Application Metadata Scheme	As defined by the Application Scheme.	As defined by the Application Metadata Scheme

The last row of the table signifies that any number of Properties may be added to Application Referenced Object instances. The only constraint is that the ULs or UUIDs values that identify the Property shall be unique within the Application Referenced Object instance.

Note 1: This means that the only Properties that cannot be added are the ones already defined to belong to the Application Referenced Object Class.

Note 2: Application Referenced Objects are encoded as MXF Local Sets. This means that, within Application Referenced Object instances, individual Properties are identified by the two byte Local Tag that is associated to the Property UL or UUID via the Primer Pack.

## Annex D Specification of the DM Plug-In Mechanism Sets (normative)

The use of these Sets is defined in Section 9.8 and illustrated in Figure 21.

### D.1 Descriptive Framework

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Descriptive Framework	Set Key	16		As defined in 9.6 (see Table 24)	Req	Defines the Superclass of all DM Frameworks	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1		See A.1	See A.1	See A.1	See A.1	
Linked Descriptive Framework Plug-In ID	Global WeakRef (DM Segment)	16	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 0c.00.00.00	Opt	Global Weak Reference to the DM Segment that strongly references this Descriptive Framework instance. Note: This is infile. [Note: SMPTE RP 210 definition Global Weak Reference to the DM Segment that strongly references this Descriptive Framework instance]	

This shall be an Abstract Class. It shall be the Superclass of all DM Frameworks.

### D.2 Descriptive Object

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Descriptive Object	Set Key	16		As defined in 9.6 (see Table 24)	Req	Defines the Superclass of all DM Objects	
All items in A.1 except the Key or Group UL and the Length, if present	See A.1		See A.1	See A.1	See A.1	See A.1	
Linked Descriptive Object Plug-In ID	Global WeakRef (DM Segment)	16	dyn	06.0E.2B.34 01.01.01.0C 05.20.07.01 11.00.00.00	Opt	Global Weak Reference to the DM Segment that indirectly strongly references this Descriptive Object instance. Note: This is infile. [Note: SMPTE RP 210 definition Global Weak Reference to the DM Segment that indirectly strongly references this Descriptive Object instance]	

This shall be an Abstract Class. It shall be the Superclass of all DM Sets.

## Annex E Specification for the Packages used in MXF (normative)

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

### E.1 Material Package

The Material Package shall be a subclass of the Generic Package defined in Annex B.1.

The Material Package shall satisfy the constraints defined in Section 9.5.3.

The following extra data values are defined:

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Material Package	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Material Package Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.1 except the Key or Group UL and the Length, if present	See B.1	See B.1	See B.1	See B.1	See B.1	See B.1	
📄	Package Marker	StrongRef (Package Marker Object)	16	dyn	06.0E.2B.34 01.01.01.0C 06.01.01.04 02.0F.00.00	Opt	A strong reference to a Package Marker Object. [Note: SMPTE RP 210 definition A strong reference to the package marker object.]	

According to Annex B.6, the Track Number Property should be set to zero (0) for all Tracks of a Material Package.

### E.2 Source Package

The Source Package shall be a subclass of the Generic Package defined in Annex B.1.

The term "Source Package" is used in this specification to mean "File Package or Physical Package".

The Source Package shall satisfy the constraints defined in Section 9.5.6.

The following extra data values are defined:

	Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
☐	Source Package	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Source Package Set	
↔	Length	BER Length	var			Req	Set length (see 9.3)	
	All items in B.1 except the Key or Group UL and the Length, if present	See B.1		See B.1	See B.1	See B.1	See B.1	
📄	Descriptor	StrongRef (EssenceDescriptor)	16	47.01	06.0E.2B.34 01.01.01.02 06.01.01.04 02.03.00.00	see E.3 and E.4	A strong reference to the Generic Descriptor (May be a Multiple Descriptor) [Note: SMPTE RP 210 definition Specifies a reference to a format descriptor for the Essence]	

Note: The Generic Descriptor and its Subclasses are defined in Annex F.

### **E.3 File Package**

The File Package shall have the structure defined in Annex E.2.

A File Package shall be associated with an Essence Container if it is referenced by a Material Package (i.e. if it is a Top-Level File Package). All other File Packages (Lower-Level File Packages) shall contain historical annotation Metadata only.

A File Package shall be identified by having one File Descriptor and zero Physical Descriptors.

Note: In AAF this Package is referred to as a File Source Package.

### **E.4 Physical Package**

The Physical Package shall have the structure detailed in Annex E.2.

A Physical Package shall not be associated with an Essence Container.

A Physical Package shall be identified by having zero File Descriptors and one Physical Descriptor.

Note: In AAF this Package is referred to as a Physical Source Package.

### **E.5 Package hierarchy in MXF**

The diagram below shows the naming conventions used in MXF.

A Material Package shall always reference Top-Level Source Packages.

A Source Package may reference Lower-Level Source Packages. They may be File Packages or Physical Packages.

Essence Container Data shall be described by a Top-Level File Package.

Lower-Level File Packages shall be used for the carriage of historical (derivation) metadata, including description of sources from which the Essence described by the Top-Level File Packages has been derived.

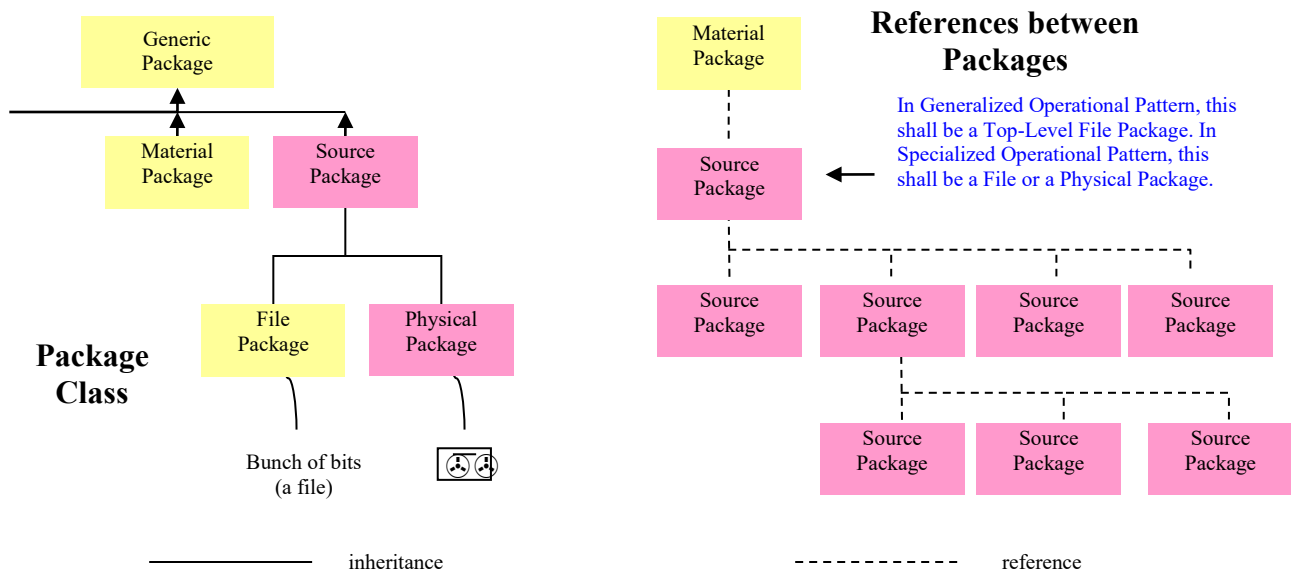


Figure E.1 – Package Class model and how Packages reference each other

## Annex F Specification of Descriptors used in MXF (normative)

This annex defines a number of Descriptors that may be implemented as KLV Sets. Other Descriptors are defined in the individual Essence Container documents where those in this Annex do not suffice.

### F.1 Scope of Descriptor Property Values

The scope of all Descriptor Properties shall be the Essence that is described by the Package that owns the Descriptor.

Example: The Properties of a Picture Essence Descriptor of a Top-Level File Package are the description of the picture inside the associated Essence Element.

### F.2 File Descriptor

This Descriptor is the Superclass from which many Descriptor sets are derived. It shall be a subclass of the Generic Descriptor that is defined in Annex B.2. Other Descriptors are defined in the individual Essence Container documents where those in this annex do not suffice. The Properties below shall be common to all Descriptor sets derived from File Descriptor.

MXF Encoders should encode a File Descriptor for each Essence Track in the File Package. MXF Encoders may omit D/req or Opt. Properties of concrete File Descriptor subclasses.

Note 1: The required column in the tables below indicates the status of Descriptor Items. Some devices implementing certain Essence Container types could be unable to fill in the Best Effort items in the Essence Descriptor at the point of file creation. For this reason the “Incomplete” Partition status exists (Section 5.2.3).

If the value of a Best Effort Metadata Item is not known by the MXF Encoder then the Distinguished Value shall be used.

Note 2: The use of the Distinguished Value requires the Partition that contains the Header Metadata to be signaled as Incomplete (Section 6.2.3).

The LinkedTrackID Property links individual File Descriptor Objects (or instances of File Descriptor Subclasses) with a specific Track in the current Package (i.e. the Package that contains the File Descriptor Object). LinkedTrackID shall be specified in each File Descriptor that is strongly referenced from a MultipleDescriptor. LinkedTrackID should not be specified in a File Descriptor that is not strongly referenced from a MultipleDescriptor (because there should be no ambiguity as to what the File Descriptor Object is describing).

In MXF, a File Descriptor (or one of its Subclasses) inside the Top-Level File Package describes how the Essence Container Data **is** coded. A File Descriptor (or one of its Subclasses) within an Lower-Level Source Package describes how a previous version of the Essence **was** coded.

The data type of certain Item ULs is itself a UL which is an enumeration of known values for the Item. The SMPTE registry that constitutes the normative reference for these values is defined by SMPTE ST 400. The values are listed in SMPTE RP 224.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
File Descriptor	Set Key	16		As defined in 9.6 (see Table 19)	Req	Defines the File Descriptor Set	
All items in B.2. except the Key or Group UL and Length, if present	See B.2.		see	See B.2.	see	See B.2.	
Linked Track ID	UInt32 (Track ID)	4	30.06	06.0E.2B.34 01.01.01.05 06.01.01.03 05.00.00.00	Opt	Link to (i.e. value of) the Track ID of the Track in this Package to which the Descriptor applies. [Note: SMPTE RP 210 definition Link to (i.e. value of) the Track ID of the Track in this Package to which the Essence Descriptor applies.]	
Sample Rate	Rational	8	30.01	06.0E.2B.34 01.01.01.01 04.06.01.01 00.00.00.00	Req	The rate of non-divisible, contiguously accessible units of the byte stream of an Essence Element (not the Essence (Pixel) sampling clock rate) [Note: SMPTE RP 210 definition Specifies the number of addressable elements of essence data per second]	
Container Duration	Length	8	30.02	06.0E.2B.34 01.01.01.01 04.06.01.02 00.00.00.00	Opt	Duration of Essence Container (measured in Edit Units) A file writer should write the best value it can write. If it cannot be completed, the Item should be omitted. [Note: SMPTE RP 210 definition Specifies the number of addressable elements of essence data]	
Essence Container	UL	16	30.04	06.0E.2B.34 01.01.01.02 06.01.01.04 01.02.00.00	Req	The UL identifying the Essence Container described by this Descriptor. Listed in SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Specifies a reference to the format of Container of Essence Data]	
Codec	UL	16	30.05	06.0E.2B.34 01.01.01.02 06.01.01.04 01.03.00.00	Opt	UL to identify a codec compatible with this Essence Container. Listed in SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Specifies a reference to the codec used to create Essence Data]	

This shall be an Abstract Class.

The value of Sample Rate shall be the rate of non-divisible, contiguously accessible units of the byte stream of an Essence Element. Essence Container specifications may define this term more strictly for individual Essence types

Examples:

1. For PCM audio, the value of Sample Rate equals the audio sampling rate.
2. For MPEG encoded audio, the value of Sample Rate equals the audio frame rate.






3. For AES3 audio carried in SMPTE ST 331 8-Channel AES3 Elements, the value of Sample Rate equals rate of 8-Channel AES3 Elements.
4. For DV-based encoded audio and video, the value of Sample Rate equals the video frame rate.
5. For MPEG-2 encoded, interlaced video applying frame coding, the value of Sample Rate equals the video frame rate.
6. For MPEG-2 encoded video applying field coding for all frames in the Essence Container, the value of Sample Rate equals the video field rate.

Note 3: Codec UL identifies the codec tool (as hardware/software) that was used to generate the Essence Container whereas the Essence Container identifies the Essence Container itself.

Note 4: The codec tool might be multi-format (e.g. MPEG/JPEG/DV). The EssenceContainers Batch in the Partition Pack contains a list of all the Essence Container as defined in Section 7.1.

### F.3 Multiple Descriptor

The Multiple Descriptor shall be a File Descriptor. It provides the method to link each Essence Track in an interleaved or compound Essence Container to the appropriate Essence Descriptor.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 Multiple Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Multiple Descriptor Set	
 Length	BER Length	var			Req	Set length (see 9.3)	
All items in F.2 except the Key or Group UL and Length, if present	See F.2		See F.2	See F.2	See F.2	See F.2	
 SubDescriptor UIDs	Array of StrongRef (File Descriptors)	8+ 16n	3F.01	06.0E.2B.34 01.01.01.04 06.01.01.04 06.0B.00.00	Req	Array of strong references to File Descriptor sets (1 per interleaved item within the Essence Container The order of the descriptors should be the same as the order of the Tracks that they describe) <b>[Note: SMPTE RP 210 definition Specifies a vector of an ordered set of references to File Descriptor sets]</b>	

Note 1: The Multiple Descriptor defines a list of descriptors which describe an Interleaved Essence Container. For example, an Essence Container which has video data interleaved with teletext data can use a Multiple Descriptor to describe each Essence type.

Note 2: Section F.2 specifies that the optional LinkedTrackID Property is not used in the Multiple Descriptor.

### F.4 Picture Essence Descriptors

#### F.4.1 Generic Picture Essence Descriptor

The Generic Picture Essence Descriptor shall be a Subclass of File Descriptor. It is designed to provide generic parametric information which describes the Picture Essence.



Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Picture Essence Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Picture Essence Descriptor Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in F.2 except the Key or Group UL and Length, if present	See F.2		See F.2	See F.2	See F.2	See F.2	
Signal Standard	Enum	1	32.15	06.0E.2B.34 01.01.01.05 04.05.01.13 00.00.00.00	Opt	Underlying Signal Standard (see G.2.3) [Note: SMPTE RP 210 definition Underlying Signal Standard]	0
Frame Layout	UInt8	1	32.0C	06.0E.2B.34 01.01.01.01 04.01.03.01 04.00.00.00	B.Effort	Interlace or progressive layout (see G.2.1) Distinguished Value = 255 (0= full_frame, 1= separate_fields, 2= single_field, 3= mixed_fields, 4=segmented_frame) [Note: SMPTE RP 210 definition Specifies frame layout (interlaced, single frame, full frame, etc.)]	
Stored Width	UInt32	4	32.03	06.0E.2B.34 01.01.01.01 04.01.05.02 02.00.00.00	B.Effort	Horizontal Size of stored picture (see G.2.6) Distinguished Value = zero (0) [Note: SMPTE RP 210 definition Specifies the integer width of the stored image in pixels]	
Stored Height	UInt32	4	32.02	06.0E.2B.34 01.01.01.01 04.01.05.02 01.00.00.00	B.Effort	Vertical Field Size of stored picture (see G.2.7) Distinguished Value = zero (0) [Note: SMPTE RP 210 definition Specifies the integer height of the stored image in pixels]	
StoredF2Offset	Int32	4	32.16	06.0E.2B.34 01.01.01.05 04.01.03.02 08.00.00.00	Opt	Topness Adjustment for stored picture (see G.2.18) [Note: SMPTE RP 210 definition Topness Adjustment for stored picture]	0
Sampled Width	UInt32	4	32.05	06.0E.2B.34 01.01.01.01 04.01.05.01 08.00.00.00	Opt	Sampled width supplied to codec (see G.2.8) [Note: SMPTE RP 210 definition Specifies the integer width of the sampled image in pixels]	Stored Width
Sampled Height	UInt32	4	32.04	06.0E.2B.34 01.01.01.01 04.01.05.01 07.00.00.00	Opt	Sampled height supplied to codec (see G.2.9) [Note: SMPTE RP 210 definition Specifies the integer height of the sampled image in pixels]	Stored Height
SampledXOffset	Int32	4	32.06	06.0E.2B.34 01.01.01.01 04.01.05.01 09.00.00.00	Opt	Offset from stored to sampled width (see G.2.10) (positive means additional stored Pixels) [Note: SMPTE RP 210 definition Specifies the X offset of the sampled image relative to the stored image in pixels]	0
SampledYOffset	Int32	4	32.07	06.0E.2B.34 01.01.01.01 04.01.05.01 0A.00.00.00	Opt	Offset from stored to sampled height (see G.2.11) (positive means additional stored lines) [Note: SMPTE RP 210 definition Specifies the Y offset of the sampled image relative to the stored image in pixels]	0

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
DisplayHeight	UInt32	4	32.08	06.0E.2B.34 01.01.01.01 04.01.05.01 0B.00.00.00	Opt	Displayed height placed in Production Aperture (see G.2.14)  [Note: SMPTE RP 210 definition Specifies the height of the presented image relative to the sampled image in pixels]	Sampled Height
DisplayWidth	UInt32	4	32.09	06.0E.2B.34 01.01.01.01 04.01.05.01 0C.00.00.00	Opt	Displayed width placed in Production Aperture (see G.2.13)  [Note: SMPTE RP 210 definition Specifies the width of the presented image in pixels]	Sampled Width
DisplayXOffset	Int32	4	32.0A	06.0E.2B.34 01.01.01.01 04.01.05.01 0D.00.00.00	Opt	Offset from SampledWidth to DisplayWidth (see G.2.15)  [Note: SMPTE RP 210 definition Specifies the X offset of the presented image relative to the sampled image in pixels]	0
DisplayYOffset	Int32	4	32.0B	06.0E.2B.34 01.01.01.01 04.01.05.01 0E.00.00.00	Opt	Offset from Sampled Height to Display Height (see G.2.16)  [Note: SMPTE RP 210 definition Specifies the Y offset of the presented image relative to the sampled image in pixels]	0
DisplayF2Offset	Int32	4	32.17	06.0E.2B.34 01.01.01.05 04.01.03.02 07.00.00.00	Opt	Topness Adjustment for displayed picture (see G.2.17)  [Note: SMPTE RP 210 definition Topness Adjustment for displayed picture]	0
Aspect Ratio	Rational	8	32.0E	06.0E.2B.34 01.01.01.01 04.01.01.01 01.00.00.00	B.Effort	Specifies the horizontal to vertical aspect ratio of the whole image as it is to be presented to avoid geometric distortion (and hence includes any black edges) e.g. {4,3} or {16,9} (see G.2.4) Distinguished Value = {0,0}  [Note: SMPTE RP 210 definition Specifies the horizontal to vertical aspect ratio of the whole image as it is to be presented to avoid geometric distortion and hence including any black edges.]	
Active Format Descriptor	UInt8	1	32.18	06.0E.2B.34 01.01.01.05 04.01.03.02 09.00.00.00	Opt	Specifies the intended framing of the content within the displayed image (4:3 in 16:9 etc.) (see G.2.5)  [Note: SMPTE RP 210 definition Specifies the intended framing of the content within the displayed image (4:3 in 16:9 etc.)]	unknown
Video Line Map	Array of Int32	8+8	32.0D	06.0E.2B.34 01.01.01.02 04.01.03.02 05.00.00.00	B.Effort	First active line in each field e.g. {16,278} (see G.2.12) Distinguished Value = {0,0}  [Note: SMPTE RP 210 definition Specifies the line numbers of the two top lines of the active picture]	
Alpha Transparency	UInt8	1	32.0F	06.0E.2B.34 01.01.01.02 05.20.01.02 00.00.00.00	Opt	Signals inversion of the Alpha component (see G.2.20).  [Note: SMPTE RP 210 definition Zero if the minimum value of an alpha sample specifies full transparency and the maximum value specifies full opacity, one if vice versa.]	0

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Transfer Characteristic	UL	16	32.10	06.0E.2B.34 01.01.01.02 04.01.02.01 01.01.02.00	Opt	Specifies the opto-electric transfer characteristic (see G.2.21). Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224) [Note: SMPTE RP 210 definition Specifies the non-linear relationship between linear scene light levels and amplitude-compressed video signal levels at signal origination.]	un-specified
Image Alignment Offset	UInt32	4	32.11	06.0E.2B.34 01.01.01.02 04.18.01.01 00.00.00.00	Opt	Byte alignment of Edit Units of stored Essence within the address space of the file (see G.2.22) [Note: SMPTE RP 210 definition Specifies number of bytes to align the start of an image with a defined memory boundary]	1
Image Start Offset	UInt32	4	32.13	06.0E.2B.34 01.01.01.02 04.18.01.02 00.00.00.00	Opt	Unused bytes before start of stored data (see G.2.23) [Note: SMPTE RP 210 definition Specifies bytes of fill before start of field]	0
Image End Offset	UInt32	4	32.14	06.0E.2B.34 01.01.01.02 04.18.01.03 00.00.00.00	Opt	Unused bytes after end of stored data (see G.2.24) [Note: SMPTE RP 210 definition Specifies bytes of fill after end of field]	0
FieldDominance	UInt8	1	32.12	06.0E.2B.34 01.01.01.02 04.01.03.01 06.00.00.00	Opt	The number of the field which is considered temporally to come first. (see G.2.19) [Note: SMPTE RP 210 definition Specifies whether the first frame of picture is field 1 or field 2]	1
Picture Essence Coding	UL	16	32.01	06.0E.2B.34 01.01.01.02 04.01.06.01 00.00.00.00	D/req	UL identifying the Picture Compression Scheme (see G.2.25) Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224). ) The value for the UL should not contain an RP 224 node. [Note: SMPTE RP 210 definition Specifies the Compression scheme used]	un-specified
Coding Equations	UL	16	32.1A	06.0E.2B.34 01.01.01.02 04.01.02.01 01.03.01.00	Opt	Specifies the encoding equations to convert RGB image components to component color difference image components. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224)	un-specified
Color Primaries	UL	16	32.19	06.0E.2B.34 01.01.01.09 04.01.02.01 01.06.01.00	Opt	Specifies the color primaries. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224)	un-specified

## F.4.2 CDCI (Color Difference Component Image) Picture Essence Descriptor

The CDCI Picture Essence Descriptor shall be a Subclass of the Generic Picture Essence Descriptor. It has all items of the Generic Picture Essence Descriptor with the same required / optional status with the addition of the new items below. It is intended to describe imagery comprising interleaved luma and color difference samples.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 CDCI Picture Essence Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the CDCI Picture Essence Descriptor Set	
 Length	BER Length	var			Req	Set length (see 9.3)	
All items in F.4.1 except the Key or Group UL and Length, if present	See F.4.1		See F.4.1	See F.4.1	See F.4.1	See F.4.1	
Component Depth	UInt32	4	33.01	06.0E.2B.34 01.01.01.02 04.01.05.03 0A.00.00.00	B.Effort	Number of active bits per sample (e.g. 8, 10, 16) or floating point format (see G.2.26) Distinguished Value = zero (0) <b>[Note: SMPTE RP 210 definition</b> Specifies the component width before subsampling is applied]	
Horizontal Subsampling	UInt32	4	33.02	06.0E.2B.34 01.01.01.01 04.01.05.01 05.00.00.00	B.Effort	Specifies the H color subsampling (see G.2.27) Distinguished Value = zero (0) <b>[Note: SMPTE RP 210 definition</b> Specifies ratio of luminance subsampling to color difference subsampling in horizontal direction]	
Vertical Subsampling	UInt32	4	33.08	06.0E.2B.34 01.01.01.02 04.01.05.01 10.00.00.00	Opt	Specifies the V color subsampling (see G.2.28) <b>[Note: SMPTE RP 210 definition</b> Specifies ratio of luminance subsampling to color difference subsampling in vertical direction]	1
Color Siting	UInt8	1	33.03	06.0E.2B.34 01.01.01.01 04.01.05.01 06.00.00.00	Opt	Enumerated value describing color siting (see G.2.29) <b>[Note: SMPTE RP 210 definition</b> Specifies how to compute subsampled color difference values]	ffh
ReversedByteOrder	Boolean	1	33.0B	06.0E.2B.34 01.01.01.05 03.01.02.01 0A.00.00.00	Opt	a FALSE value denotes color difference followed by luma samples according to ITU Rec.601 (see G.2.35) <b>[Note: SMPTE RP 210 definition</b> Specifies whether the luma and chroma sampling order conforms to ITU-R BT.601. Value will be zero if the byte order conforms, non-zero if the luminance sample precedes the chroma.]	false

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
PaddingBits	Int16	2	33.07	06.0E.2B.34 01.01.01.02 04.18.01.04 00.00.00.00	Opt	Bits to round up each pixel to stored size (see G.2.30)  [Note: SMPTE RP 210 definition Specifies the number of bits to pad each pixel so that the next pixel starts on a defined boundary]	0
Alpha Sample Depth	UInt32	4	33.09	06.0E.2B.34 01.01.01.02 04.01.05.03 07.00.00.00	Opt	Number of bits per alpha sample (see G.2.31)  [Note: SMPTE RP 210 definition Specifies the number of bits in the alpha signal.]	0
Black Ref Level	UInt32	4	33.04	06.0E.2B.34 01.01.01.01 04.01.05.03 03.00.00.00	Opt	e.g. 16 or 64 (8 or 10-bits) (see G.2.32)  [Note: SMPTE RP 210 definition Specifies digital luminance associated with black]	0
White Ref level	UInt32	4	33.05	06.0E.2B.34 01.01.01.01 04.01.05.03 04.00.00.00	Opt	e.g. 235 or 940 (8 or 10 bits) (see G.2.33)  [Note: SMPTE RP 210 definition Specifies digital luminance associated with white]	maximum unsigned integer value for the component size
Color Range	UInt32	4	33.06	06.0E.2B.34 01.01.01.02 04.01.05.03 05.00.00.00	Opt	e.g. 225 or 897 (8 or 10 bits) (see G.2.34)  [Note: SMPTE RP 210 definition Specifies the range of the color levels.]	maximum unsigned integer value for the component size

### F.4.3 RGBA (Red Green Blue Alpha) Picture Essence Descriptor

The RGBA Picture Essence Descriptor shall be a Subclass of the Generic Picture Essence Descriptor above. It has all items of the Generic Picture Essence Descriptor with the same required / optional status with the addition of the new items below. It is intended to describe color component imagery with transparency.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
RGBA Picture Essence Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the RGBA Essence Descriptor Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in F.4.1 except the Key or Group UL and Length, if present	See F.4.1		See F.4.1	See F.4.1	See F.4.1	See F.4.1	
Component Max Ref	UInt32	4	34.06	06.0E.2B.34 01.01.01.05 04.01.05.03 0B.00.00.00	Opt	Maximum value for RGB components, e.g. 235 or 940 (8 or 10 bits) [Note: SMPTE RP 210 definition Maximum value for RGB components e.g. 235 or 940 (8 or 10 bits)]	255
Component Min Ref	UInt32	4	34.07	06.0E.2B.34 01.01.01.05 04.01.05.03 0C.00.00.00	Opt	Minimum value for RGB components, e.g. 16 or 64 (8 or 10-bits) [Note: SMPTE RP 210 definition Minimum value for RGB components e.g. 16 or 64 (8 or 10-bits)]	0
Alpha Max Ref	UInt32	4	34.08	06.0E.2B.34 01.01.01.05 04.01.05.03 0D.00.00.00	Opt	Maximum value for alpha component, e.g. 235 or 940 (8 or 10 bits) [Note: SMPTE RP 210 definition Maximum value for alpha component e.g. 235 or 940 (8 or 10 bits)]	255
Alpha Min Ref	UInt32	4	34.09	06.0E.2B.34 01.01.01.05 04.01.05.03 0E.00.00.00	Opt	Minimum value for alpha components, e.g. 16 or 64 (8 or 10-bits) [Note: SMPTE RP 210 definition Minimum value for alpha components e.g. 16 or 64 (8 or 10-bits)]	0
ScanningDirection	Orientation	1	34.05	06.0E.2B.34 01.01.01.05 04.01.04.04 01.00.00.00	Opt	Enumerated Scanning Direction (see G.2.39) [Note: SMPTE RP 210 definition Enumerated Scanning Direction]	0
PixelLayout	RGBALayout	16	34.01	06.0E.2B.34 01.01.01.02 04.01.05.03 06.00.00.00	B.Effort	(see G.2.36) Distinguished Value = zero (0) in each byte of the RGBALayout Array [Note: SMPTE RP 210 definition Specifies pixel quantization and order as a data structure.]	
Palette	DataValue	var	34.03	06.0E.2B.34 01.01.01.02 04.01.05.03 08.00.00.00	Opt	(see G.2.37) [Note: SMPTE RP 210 definition Specifies, as a single string, the fixed length values of each color in the palette used.]	
PaletteLayout	RGBALayout	var	34.04	06.0E.2B.34 01.01.01.02 04.01.05.03 09.00.00.00	Opt	(see G.2.38) [Note: SMPTE RP 210 definition Specifies pixel quantization and order in the palette as a data structure.]	

## F.5 Generic Sound Essence Descriptor

The Generic Sound Essence Descriptor shall be a Subclass of the File Descriptor. It is designed to provide generic parametric information which describes the Sound Essence.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Sound Essence Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Sound Essence Descriptor Set	
Length	BER Length	var			Req	Set length (see 9.3)	
All items in F.2 except the Key or Group UL and Length, if present	See F.2		See F.2	See F.2	See F.2	See F.2	
Audio sampling rate	Rational	8	3D.03	06.0E.2B.34 01.01.01.05 04.02.03.01 01.01.00.00	B.Effort	Sampling rate of the audio Essence Distinguished Value = (0, 1) [Note: SMPTE RP 210 definition The reference sampling clock frequency as a rational number]	
Locked/Unlocked	Boolean	1	3D.02	06.0E.2B.34 01.01.01.04 04.02.03.01 04.00.00.00	D/req	Boolean indicating that the number of samples per frame is locked or unlocked. [Note: SMPTE RP 210 definition TRUE if number of samples per frame is locked to video]	
Audio Ref Level	Int8	1	3D.04	06.0E.2B.34 01.01.01.01 04.02.01.01 03.00.00.00	Opt	Audio reference level which gives the number of dBm for 0VU. [Note: SMPTE RP 210 definition Number of dBm for 0VU]	
Electro-Spatial Formulation	UInt8 (Enum)	1	3D.05	06.0E.2B.34 01.01.01.01 04.02.01.01 01.00.00.00	Opt	E.g. mono, dual mono, stereo, A,B etc [Note: SMPTE RP 210 definition Mono, Dual mono, Stereo A+B, Stereo M&S, Dolby surround, MPEG BC/NBC etc] 0 = two-channel mode default 1 = two-channel mode 2 = single channel mode 3 = primary/secondary mode 4 = stereophonic mode 7 = single channel, double frequency mode carried on 2 sub-frames 8 = stereo left channel, double frequency mode carried on 2 sub-frames 9 = stereo right channel, double frequency mode carried on 2 sub-frames 15 = multi-channel mode default (>2 channels) Note: These values are identical to values defined in AES3.	0
ChannelCount	UInt32	4	3D.07	06.0E.2B.34 01.01.01.05 04.02.01.01 04.00.00.00	B.Effort	Number of sound channels Distinguished Value = zero (0) [Note: SMPTE RP 210 definition The number of channels represented in the waveform data.]	
Quantization bits	UInt32	4	3D.01	06.0E.2B.34 01.01.01.04 04.02.03.03 04.00.00.00	B.Effort	Number of quantization bits Distinguished Value = zero (0) [Note: SMPTE RP 210 definition The maximum number of significant bits for the value without compression.]	



Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
Dial Norm	Int8	1	3D.0C	06.0E.2B.34 01.01.01.05 04.02.07.01 00.00.00.00	Opt	Gain to be applied to normalize perceived loudness of the clip, defined by ITU-R BS.1196-2 2010 (1dB per step) <b>[Note: SMPTE RP 210 definition</b> Gain to be applied to normalize perceived loudness of the clip] (Defined by ITU-R BS.1196-2 2010 (1dB per step)	
Sound Essence Coding	UL	16	3D.06	06.0E.2B.34 01.01.01.02 04.02.04.02 00.00.00.00	D/req	UL identifying the Sound Compression Scheme. Individual UL values are listed in the Registry defined by SMPTE ST 400 (RP 224). The value for the UL should not contain an RP 224 node. <b>[Note: SMPTE RP 210 definition</b> Specifies the Compression scheme used]	



If the Sound Essence Coding Property is not present in the Descriptor, MXF decoders shall assume the SMPTE ST 400 (RP 224) Label Designator value or 06.0E.2B.34.04.01.01.0A and the SMPTE ST 400 (RP 224) Item Designator value of 04.02.02.01.01.00.00.00.

Note 1: This corresponds to the case of uncompressed sound Essence that employs the same bit-by-bit packing of sample data as is used by AES31-2 or EBU Tech-3285 (BWF).

Note 2: SMPTE ST 382 defines Subclasses of the Generic Sound Essence Descriptor that are suitable for describing PCM and AES Essence Elements.

## F.6 Generic Data Essence Descriptor

The Generic Data Essence Descriptor shall be a Subclass of the File Descriptor. It is designed to provide generic parametric information which describes the Data Essence.

Item Name	Type	Len	Local Tag	Item UL	Req ?	Meaning	Default
 Data Essence Descriptor	Set Key	16		As defined in 9.6 (see Table 17)	Req	Defines the Data Essence Descriptor Set	
 Length	BER Length	Var			Req	Set length (see 9.3)	
All items in F.2 except the Key or Group UL and Length, if present	See F.2		See F.2	See F.2	See F.2	See F.2	
Data Essence Coding	UL	16	3E.01	06.0E.2B.34 01.01.01.03 04.03.03.02 00.00.00.00	D/req	Specifies the data Essence coding type Values are listed in SMPTE ST 400 (RP 224). The value for the UL should not contain an RP 224 node. <b>[Note: SMPTE RP 210 definition</b> Specifies the Coding scheme used]	

Note: It is anticipated that many Data Descriptors will need to be created for MXF. These are likely to be used to describe data buried within other Essence types (e.g. timecode, teletext, subtitles, captioning) as well as new data types that MXF will carry in the future. A document which defines the Container Specification for the data Essence will define or reference Data Descriptors and the appropriate Data Definition term to be used in the Sequence and Segment.

Annex G Picture Essence Descriptor Properties (normative)

The Picture Essence Descriptor Properties are appropriate to describe progressive and interlaced pictures where the setting of a Property depends on the picture storage format (i.e. progressive frame, interlaced frame, separated fields of an interlaced frame, a progressive frame stored as segmented frames (see Annex G.2.1)). Where relevant, the precise values for each of the storage formats are defined in the following subsections.

Frame-based compression of interlaced pictures shall always be described as if a progressive frame, unless overwritten by an Essence Container specification.

Optional Picture Descriptor Properties shall only be encoded in the MXF file if their correct value is known to the MXF encoder. They shall not be encoded in any other case.

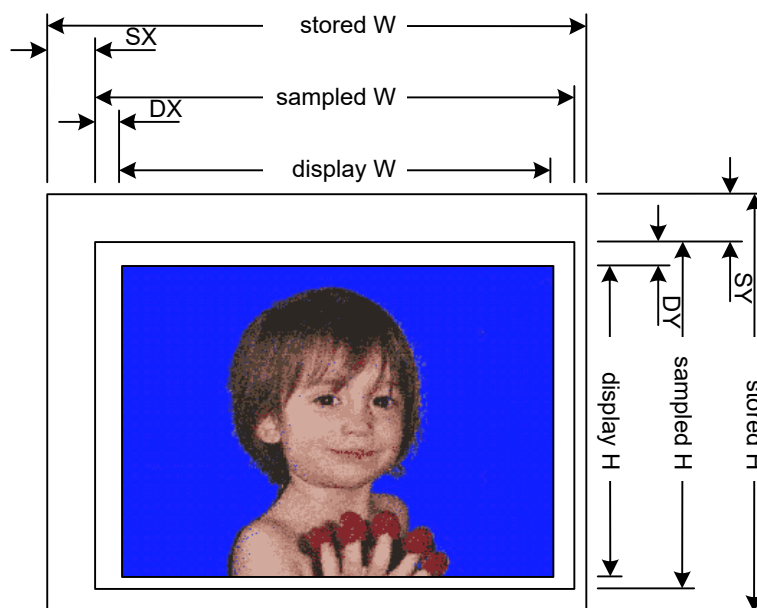
The Picture Descriptor Properties defined in this section describe the Picture Essence that is associated with the Package that owns the Descriptor. They do not describe scaling or subsampling operations.

G.1 Data Storage, Sampling, Display and Video Interface

This specification describes image geometries from three perspectives: the Stored Rectangle, the Sampled Rectangle and the Display Rectangle.

The specification defines the image properties, plus properties to describe the mapping of image rectangles between each view and to the actual video interface.





Legend:

SX - sampled X offset

SY - sampled Y offset

DX - display X offset

DY - display Y offset

**Figure G.1 – Stored, Sampled, Display Rectangles and Video Interface**

### G.1.1 Stored Data and Stored Rectangle

The Stored Data shall comprise the entire data region corresponding to the Stored Rectangle for a single frame or field of the image plus any start or end data bytes in the byte stream. The Stored Rectangle shall correspond to a rectangle of stored Pixels described by the Stored Width and Stored Height Properties. The Stored Rectangle byte count shall be defined by the product of its width, its height and the number of bytes per Pixel or by the number of bytes in the corresponding compressed bit stream.

The Stored Data may include bytes that are not derived from, and would not usually be translated back to, signal data. This extra data is the Start Fill and End Fill region. The sizes shall be defined by the Image Start Offset and Image End Offset Properties.

The Stored Data may be aligned to a particular storage boundary, defined by the Image Alignment Offset.

Note: In cases where compression systems are in use, the dimensions of the Stored Rectangle need to be set to the dimensions of the rectangle passed to the picture essence encoder and returned by the picture essence decoder, including any decoder scaling or padding. Since compression systems typically constrain the sizes of macroblocks, the Stored Rectangle can be larger than the Sampled Rectangle and the Display Rectangle. Also, in these cases, the definition of the macroblocks can dictate an offset from Stored to Sampled Rectangle which is not equal in field 1 and field 2; this is described by the optional StoredF2Offset Property below. In addition, when the Stored Rectangle comprises merged fields, the Sample Rate Property will be set to the frame rate according to Annex G.2.2.

### G.1.2 Sampled Rectangle

The Sampled Rectangle shall be the rectangular region corresponding to the digital data Pixels derived from an image source. It shall include the image and any auxiliary information actually sampled from the analog or digital source.

The Sampled Rectangle data shall be the same size or a subset of the Stored Rectangle.

The Sampled Rectangle shall be defined by its Width and Height Properties in Pixels. These Properties shall default to the same value as the Stored Width and Height. Their values shall not be greater than the Stored Width and Stored Height.

The mapping from the Stored Rectangle to the Sampled Rectangle shall be defined by the SampledXOffset (SX) and SampledYOffset (SY) Properties, which give the zero-based coordinates of the Sampled Rectangle relative to the upper left corner of the Stored Rectangle. These Properties shall have a default value of zero (0). Their values shall not be negative.

Note: The example in Figure G.1 shows positive values for the SX and SY Offsets.

### G.1.3 Display Rectangle

The Display Rectangle shall be the rectangular region which is visible in the display device. It shall not include VBI lines. It shall not include extra lines included into the sampled rectangle to extend the image to satisfy the size requirements of the compression system.

The Display Rectangle shall be defined by its Width and Height Properties in Pixels. These Properties shall default to the same value as the Sampled Width and Sampled Height. The Display Rectangle Width and Height values shall be no greater than the Sampled Width and Sampled Height values taking into account any Display X Offset (DX) and Display Y Offset (DY) Property values.

The mapping from the Sampled Rectangle to the Display Rectangle shall be defined by the Display X Offset (DX) and Display Y Offset (DY) Properties, which give the zero-based coordinates of the Display Rectangle relative to the upper left corner of the Sampled Rectangle. These Properties shall have a default value of zero (0). Their values shall not be negative.

### G.1.4 Video Interface

The Video Interface parameters are provided so that pictures that have originated from a source video interface may be accurately placed back onto the same Video Interface. These parameters may be present whether or not the picture has been compressed.

The value of the Video Interface shall constitute a reference to Video Interface such as by SMPTE ST 125, SMPTE ST 274, SMPTE ST 296, ITU-R BT.470-7, and ITU-R BT.601-7. Companion standards such as ITU-R BT.656-5 give additional description for some cases.

Images may be scanned in progressive or interlaced mode. This is specified by the Frame Layout Property.

Different standards define different line numbering schemes. The cardinal Line Numbers were initially specified for analogue standards, such as ITU-R BT.470. The definitions of fields and line numbers in digital images are not identical to the definitions for analogue images. The actual numbers used by this specification are those defined for the digital standards (ITU-R BT.656, SMPTE ST 274, SMPTE ST 296, SMPTE ST 293).

The Video Interface provides the framework in which the Display Rectangle (Production Aperture) is positioned. The vertical mapping of the Display Rectangle into the Video Interface shall be defined in three stages:

1. by the VideoLineMap Property, which shall specify the cardinal line numbers of the first sampled line in each field of the Video Interface

2. by the DisplayYOffset Property, which shall specify the number of lines of sampled data which must be blanked at the start of each field
3. by an optional DisplayF2Offset Property, which shall adjust the number of lines which are blanked in field 2 relative to the number blanked in field 1

The horizontal mapping of the Display Rectangle into the Video Interface shall be defined by blanking the Pixels to the left of DisplayXOffset and to the right of DisplayXOffset+DisplayWidth-1.

### **G.1.5 Sampling**

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

#### **G.1.5.1 Color Difference Component Sampling (CDCI Sampling)**

Color Difference Component Sampling is the method used by ITU-R BT.601-7, SMPTE ST 125, SMPTE ST 274, SMPTE ST 296, and other standards. These standards define color difference components which may be subsampled.

CDCI imagery shall be described by a CDCI Picture Essence Descriptor or a Subclass. This Descriptor contains Properties appropriate for this type of imagery.

MXF wrappings of imagery compressed according to specifications such as MPEG-2 and DV use the CDCI Picture Essence Descriptor or a Subclass of the CDCI Picture Essence Descriptor.

The CDCI Picture Essence Descriptor can describe imagery with subsampled lattices such as 4:4:4, 4:2:2, 4:2:0, 4:1:1 lattices that have the same number of bits for each component.

#### **G.1.5.2 Red Green Blue Alpha Sampling (RGBA Sampling)**

When Red Green Blue Alpha Sampling is used, each Pixel shall be composed of co-sited samples from Red, Green, Blue and, optionally, Alpha channels. The complete list of color space representations that may be used are defined in Annex G.2.36.

RGBA imagery shall be described by an RGBA Picture Essence Descriptor or a Subclass.

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

### **G.2 Property Definitions**

This section specifies the Properties which are used by the Picture Descriptors. Some of these Properties are described as Inferred or Derived:

**Inferred** Properties are those whose values may be easily inferred from the underlying standard (as given by the Signal Standard Property or by calculation from the values of other Properties).

**Derived** Properties are useful parameters which can be derived from other Properties.

#### **G.2.1 Frame Layout**

Images may be scanned progressively or in one of several interlaced methods.

The Frame Layout Property shall identify the scanning as defined in the following sections.

Note: Together with the Sample Rate Property, the FrameLayout Property provides the information from which SMPTE ST 352 byte 2 may be derived.

#### **G.2.1.1 FULL\_FRAME**

This shall be indicated by a value of zero (0).

A progressive lattice from top to bottom, stored in progressive line order 1,2,3,4,5,6... The duration of a Sampled Rectangle shall be a Frame

Example: “480P59.94”.

#### **G.2.1.2 SEPARATE\_FIELDS**

This shall be indicated by a value of one (1).

An interlaced lattice divided into two fields, stored as two fields 1,3,5,... and 2,4,6.... Field 1 scans alternate lines from top to bottom, field 2 scans the intervening lines. The second field is scanned at a later time than the first field (one field later). Different signal standards may define different Topness (see Annex G.2.18) and Dominance (see Annex G.2.19). The duration of a Sampled Rectangle shall be a Field.

Examples: NTSC, SMPTE ST 125.

#### **G.2.1.3 SINGLE\_FIELD**

This shall be indicated by a value of two (2).

An interlaced lattice as for SEPARATE\_FIELDS above, except that only one field is scanned and retained in the stored data, as 1,3,5,... or 2,4,6,... or (1+2),(3+4),(5+6),.... For display, the second field is derived by line replication or interpolation. The duration of a Sampled Rectangle shall be a Frame.

Examples: There are no examples of SINGLE\_FIELD in broadcast use; however, this type of sub-sampling is often used as a simple compression for index frames.

#### **G.2.1.4 MIXED\_FIELDS**

This shall be indicated by a value of three (3).

An interlaced lattice as for SEPARATE\_FIELDS above, stored as a single matrix of interleaved lines. The duration of a Sampled Rectangle shall be a Frame.

Examples: It is not common to use MIXED\_FIELDS in broadcast; however, intermediate in-memory data structures sometimes use this format.

#### **G.2.1.5 SEGMENTED\_FRAME**

This shall be indicated by a value of four (4).

An interlaced lattice divided into two fields. Field 1 scans alternate lines from top to bottom, field 2 scans the intervening lines. The lines are stored as two fields 1,3,5,... 2,4,6,.... The two fields are taken from a single scan of the incoming image — i.e., they are coincident in time, except for the effects of shutter angle. The duration of a Sampled Rectangle shall be a Field.

Essence Container specifications shall identify the Frame Layout values which shall be used by the Picture Descriptor.

Example: “1080P24 PsF”.

### G.2.2 Sample Rate and Edit Rate

Sample Rate is used in Descriptors and Edit Rate is used in Tracks. Sample Rate shall equal to the rate of Stored Rectangles expressed as a rational number. This is typically either the rate of fields or frames in the image.

The term Sample Unit is used in the text below to represent  $1 / \text{Sample Rate}$ , i.e. the time duration of the Stored Rectangle.

Edit Rate shall equal the desired editing rate of the image data for the application. A commonly used Edit Unit (i.e.  $1/\text{Edit Rate}$ ) is 1 image, but may be smaller or larger in some applications.

Note 1: For example, the sampled image may be field based, but the editing may be frame based. In this case the Edit Rate will be half the Sample Rate.

Note 2: Together with the Frame Layout Property, the Sample Rate Property provides the information from which SMPTE ST 352 byte 2 may be derived.

### G.2.3 Signal Standard

The Signal Standard enumerated value shall indicate the source underlying signal standard of the video interface from which the stored data was created, where applicable.

The default value shall be 00h.

Valid values are:

Value	Meaning
00h	No specific underlying standard
01h	ITU-R BT.601 and BT.656, also SMPTE ST 125 (525 and 625 line interlaced)
02h	ITU-R BT.1358 and ITU-R BT.799-3, also SMPTE ST 293 (525 and 625 line progressive)
03h	SMPTE ST 347 (540 Mbps mappings)
04h	SMPTE ST 274 (1125 line)
05h	SMPTE ST 296 (750 line progressive)
06h	SMPTE ST 349 (1485 Mbps mappings)
07h	SMPTE ST 428-1 DCDM

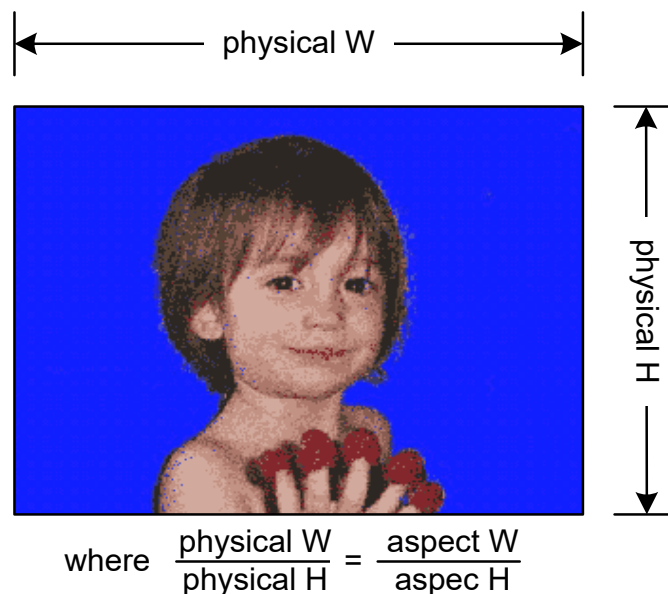
Note: This Property matches the standards defined by SMPTE ST 352 byte 1.

### G.2.4 Aspect Ratio

The Aspect Ratio shall equal the ratio of width to height of the physical representation of the Display Rectangle as a rational.

Example: {4,3}, {16,9}

Note: Not all display rectangles are constructed from square Pixels.



### G.2.5 Active Format Descriptor (AFD)

Active Format Description, as defined by SMPTE ST 2016-1, describes a video picture in terms of the aspect ratio and other characteristics of the active image within the coded frame. In general, Active Format Description can vary within a video stream, reflecting changes to the formatting of the video picture within the coded frame over time. In particular cases, Active Format Description might be constant for an individual video clip or stream.

Note 1: The SMPTE RP 210 name for the element used here in the Picture Descriptor is Active Format Descriptor, in contrast to the SMPTE ST 2016-1 term Active Format Description.

For cases where the Active Format Description is constant for the duration of the essence container (whether encoded as MXF essence elements or embedded in the essence), then the Active Format Descriptor element may be encoded in the Picture Descriptor. For cases where the Active Format Description varies with time, the Active Format Descriptor element shall not be encoded in the Picture Descriptor. If the Active Format Description is encoded in the Picture Descriptor, then it shall be encoded if and only if it is constant in the essence container (whether encoded as MXF essence elements or embedded in the essence).

If the Active Format Description is time-varying, then it should be encoded as VANC data according to SMPTE ST 2016-3 and embedded in an MXF essence container according to SMPTE ST 436.

For decoders of MXF files: if the Active Format Descriptor value is encoded in the Picture Descriptor, the essence track shall be assumed to have a constant Active Format Description value in the essence container. If the Active Format Description value is not encoded in the picture descriptor, no assumptions can be made about whether a per-frame Active Format Description value is encoded in the essence container, or whether it is constant or not.

The Active Format Descriptor element in the Picture Descriptor is expressed as an 8-bit unsigned integer (b7-b0) with characteristics as follows:



## Encoders

Encoders compliant with SMPTE 377-1 shall encode b7-b0 as defined in SMPTE ST 2016-1 Table 4, where b7, b1, b0 are each '0' (reserved), b6-b3 are the AFD code data bits a3-a0, and b2 is an aspect ratio bit.

Note 2: AFD code data bits a3-a0 are defined in SMPTE ST 2016-1 Table 1. The aspect ratio (AR) bit uses value '1' to indicate 16:9 and '0' to indicate 4:3.

## Decoders

For decoders compliant with SMPTE 377-1:

If the MXF file being read has the Version property of the Preface Set equal to 1.3 or higher:

- decoders shall ignore b7,
- decoders shall interpret b6-b3 as AFD a3a2a1a0,
- decoders shall interpret b2 as AR,
- and
- decoders shall ignore b1-b0.

If the MXF file being read has the Version property of the Preface Set lower than 1.3 and both b6 and b5 equal 0:

- decoders should ignore b7,
- decoders should interpret b3-b0 as AFD a3a2a1a0,
- and
- decoders may interpret b4 as AR.

If the MXF file being read has the Version property of the Preface Set lower than 1.3 and either b6 equals 1 or b5 equals 1:

- decoders should ignore b7,
- decoders should interpret b6-b3 as AFD a3a2a1a0,
- decoders should interpret b2 as AR,
- and
- decoders should ignore b1-b0.

AFD code data bits a3-a0 are defined in SMPTE ST 2016-1 Table 1. The aspect ratio (AR) bit uses value '1' to indicate 16:9 and '0' to indicate 4:3.

In all cases for decoders, if the Aspect Ratio property of the GenericPictureEssenceDescriptor Set is specified in the file being read, it shall take precedence over the AR bit.

Note 3: It is possible that decoders compliant with SMPTE 377-1, when reading MXF files with the Version property of the Preface Set lower than 1.3, will not correctly decode AFD codes 0000, 0010, and 0011 carried in the byte defined in SMPTE ST 2016-1 Table 4.

### G.2.6 Stored Width

Stored Width shall be equal to the number of Pixels across the Stored Rectangle, expressed as a 32-bit unsigned integer.

### G.2.7 Stored Height

Stored Height shall be equal to the number of Pixels from top to bottom of the Stored Rectangle, expressed as a 32-bit unsigned integer.

The table below defines the relationship between the Frame Layout value and the Stored Height.

Frame Layout value	Stored Height
FULL_FRAME	frame height
SEPARATE_FIELDS	field height
SINGLE_FIELD	field height
MIXED_FIELDS	frame height
SEGMENTED_FRAME	frame height

### G.2.8 Sampled Width

Sampled Width shall be equal to the number of Pixels across the Sampled Rectangle, expressed as a 32-bit unsigned integer.

The default value shall be the Stored Width value.

Sampled Height shall be equal to the number of Pixels from top to bottom of the Sampled Rectangle, expressed as a 32-bit unsigned integer.

The table below defines the relationship between the Frame Layout value and the Sampled Height.

Frame Layout value	Sampled Height
FULL_FRAME	frame height
SEPARATE_FIELDS	field height
SINGLE_FIELD	field height
MIXED_FIELDS	frame height
SEGMENTED_FRAME	frame height

The default value shall be the Stored Height value.

### G.2.10 SampledXOffset

SampledXOffset shall be equal to the horizontal offset in Pixels of the left edge of the Sampled Rectangle relative to the left edge of the Stored Rectangle, expressed as a 32-bit signed integer. The value shall be non-negative.

The default value shall be zero (0).

### G.2.11 SampledYOffset

SampledYOffset shall be equal to the vertical offset in Pixels of the upper edge of the Sampled Rectangle relative to the upper edge of the Stored Rectangle, expressed as a 32-bit signed integer. The value shall be non-negative.

The default value shall be zero (0).

### G.2.12 Video Line Map

Video interfaces define the Line Numbers of the Video Interface (for example 1-525). Only a portion of the data from the video interface is stored in the Sampled Rectangle.

Video Line Map shall specify the Line Numbers of the first line(s) in the video interface to which the Sampled Rectangle is mapped.

Note: These Line Numbers do not include zero. The first line of the interface is numbered Line 1.

The order of the two numbers shall not be used to denote Topness or Field Dominance.

Frame Layout value	Video Line Map	Example
FULL_FRAME	{first line number, zero (0)}	{26, 0} for 720p active picture from SMPTE ST 296
SEPARATE_FIELDS	{first line number first field, first line number second field}	{21,584} for 1080i active picture from SMPTE ST 274
SINGLE_FIELD	{first line number first field, first line number second field}	{21,584} for 1080i active picture from SMPTE ST 274
MIXED_FIELDS	{first line number first field, first line number second field}	{21,584} for 1080i active picture from SMPTE ST 274
SEGMENTED_FRAME	{first line number first field, first line number second field}	{21,584} for 1080 PsF active picture from SMPTE ST 274

### G.2.13 DisplayWidth

DisplayWidth shall be equal to the number of Pixels across the Display Rectangle, expressed as a 32-bit unsigned integer.

The default value shall be the value of Sampled Width.

### G.2.14 DisplayHeight

DisplayHeight shall be equal to the number of Pixels from top to bottom of the Display Rectangle, expressed as a 32-bit unsigned integer.

The table below defines the relationship between the Frame Layout value and the Display Height.

Frame Layout value	Display Height
FULL_FRAME	frame height
SEPARATE_FIELDS	field height
SINGLE_FIELD	field height
MIXED_FIELDS	frame height
SEGMENTED_FRAME	frame height

The default value shall be the Sampled Height value.

### G.2.15 DisplayXOffset

Display X Offset shall be the horizontal offset in Pixels of the left edge of the Display Rectangle relative to the left edge of the Sampled Rectangle, expressed as a 32-bit signed integer.

The default value shall be zero (0).

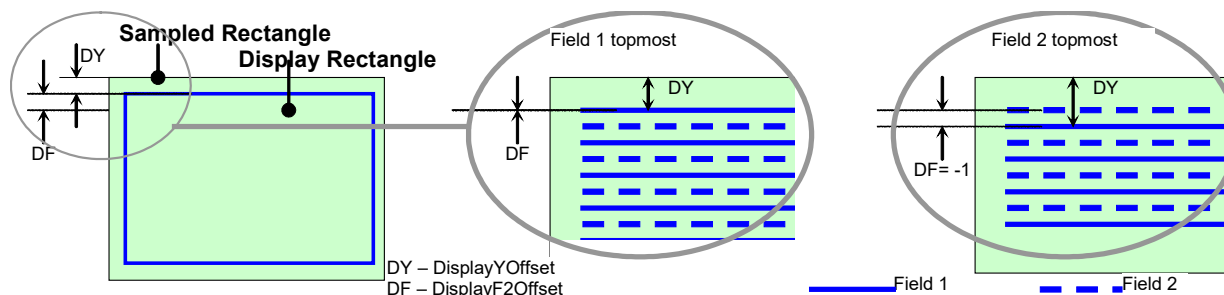
### G.2.16 DisplayYOffset

Display Y Offset shall be the vertical offset of the upper edge of the Display Rectangle relative to the upper edge of the Sampled Rectangle, expressed as a 32-bit signed integer.

The default value shall be zero (0).

### G.2.17 DisplayF2Offset

The normal relationship between Sampled and Display Rectangles is the same for both the first field and the second field. In some cases, the Sampled Rectangle for the first field starts with data from the interlaced line above the first line of the second field (i.e., Sampled Topness is first field upper), even though the Display Rectangle begins with a line from the second field (i.e., Displayed Topness is second field upper).



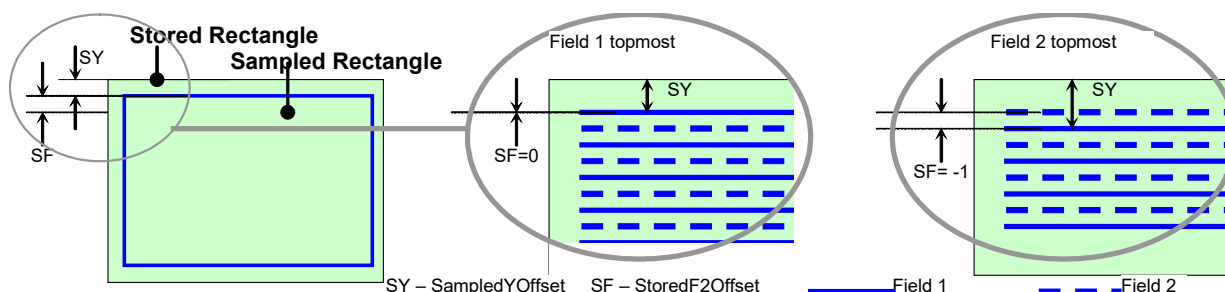
The DisplayF2Offset Property adjusts the DisplayYOffset for the second field relative to that for the first field. Its value shall be zero (0) or minus 1. A value of minus 1 shall invert the Displayed Topness relative to the Sampled Topness.

The default value shall be zero (0).

In conjunction with Frame Layout values of FULL\_RAME and SEGMENTED\_FRAME, the DisplayF2Offset Property has no meaning and should not be present.

### G.2.18 StoredF2Offset

The normal relationship between Stored and Sampled Rectangles is the same for both the first field and the second field. In some cases, the Stored Rectangle for the first field starts with data from the interlaced line above the first line of the second field (i.e., Stored Topness is first field upper), even though the Sampled Rectangle begins with a line from the second field (i.e., Sampled Topness is second field upper).



The StoredF2Offset Property adjusts the SampledYOffset for the second field relative to that for the first field. Its value shall be zero (0) or minus 1. A value of minus 1 shall invert the Sampled Topness relative to the Stored Topness.

The default value shall be zero (0).

Note: StoredF2Offset is minus 1 for MPEG-2 422P compression of 525-line video.

In conjunction with Frame Layout values of FULL\_FRAME and SEGMENTED\_FRAME, the StoredF2Offset Property has no meaning and should not be present.

## G.2.19 FieldDominance

Field Dominance is a Property, whose unsigned 8-bit integer value shall specify which field is the first field in temporal order in an interlaced frame.

A value of 1 shall indicate that the first field is first in temporal order. A value of 2 shall indicate that the second field is the first in temporal order.

The default value shall be one (1).

In conjunction with Frame Layout values of FULL\_FRAME and SEGMENTED\_FRAME, the value of field dominance has no meaning and should not be present.

## G.2.20 Alpha Transparency

Alpha Transparency is an unsigned 8-bit value. It shall have a value of zero (FALSE) if zero values of the Alpha channel represent a fully transparent Pixel. It shall have a value of one (TRUE) if zero values of the Alpha channel represent a fully opaque Pixel.

The default value shall be zero (0).

Note: This flag is a modifier to the Alpha Channel data where present as part of the Essence. Its purpose is to define which Alpha Channel value defines the Essence data to be transparent.

## G.2.21 Transfer Characteristic

Transfer Characteristic is a Property, whose value is a 16-byte Universal Label of a registered set of color primaries, color matrix and gamma equation. Values are listed in the Register defined by SMPTE ST 400 (RP 224).

The default value is not defined.

MXF encoders should encode Transfer Characteristic whenever possible.

### G.2.22 Image Alignment Offset

The Image Alignment Offset Property is deprecated.

Image Alignment Offset is a Property, whose unsigned 32-bit integer value shall specify the byte alignment of the Edit Units of stored Essence in the address space of the file (address space [0] = first byte of the KLV key of the Header Partition pack).

Example: A value of 16 specifies that image data is aligned on 16 byte boundaries.

The default value shall be one (1).

This Property is provided for compatibility with some formats which may be wrapped by MXF (for example some uncompressed disk formats). New implementations should use KAG rules and the Image Alignment Offset should be omitted unless backwards compatibility with the existing format requires a value greater than 1.

Note: Image Alignment Data will not be KLV wrapped, whereas KAG filling is KLV wrapped. For this reason, the Image Alignment Offset Property exists for backwards compatibility.

### G.2.23 Image Start Offset

Image Start Offset is a Property, whose unsigned 32 bit integer value shall specify the number of unused bytes from the start of the Stored Data for a Sample Unit to the start of the Stored Rectangle.

The default value shall be zero (0).

### G.2.24 Image End Offset

Image End Offset is a Property, whose unsigned 32 bit integer value shall specify the number of unused bytes from the end of the Stored Rectangle to the end of the Stored Data for a given Sample Unit.

The default value shall be zero (0).

### G.2.25 Picture Essence Coding

Picture Essence Coding shall be a 16-byte UL value that shall define the Essence coding / compression scheme in use. Values are listed in the Register defined by SMPTE ST 400 (RP 224).

If the value of the Picture Essence Coding Property conflicts with the value of any other Property of the CDCI Picture Essence Descriptor (i.e. Component Depth, Horizontal Subsampling, Vertical Subsampling, ReversedByteOrder, PaddingBits or Alpha Sample Depth), then value of the Picture Essence Coding Property shall take precedence.

In the case of uncompressed Essence, there are a number of sub-sampled grid layouts that cannot be expressed with either the PixelLayout in the RGBA Picture Essence Descriptor, or Properties like ReversedByteOrder and PaddingBits in the CDCI Picture Essence Descriptor. The table below is a description of component layouts that can be referenced by using certain ULs for Picture Essence Coding. The method of describing these layouts is similar to that used in the Pixel Layout Item, except that the UL from the table below is encoded in the Property, and the table describes each layout detail.

This intentionally limits the number of possible layouts that can be expressed with this Property, which in turn constrains the scope of implementation, testing and interoperability.

Note: The different labels essentially provide an encoding for the permutations of four parameters of each layout: interleave vs. planar; bit depth; four color subsamplings (listed below); and six possible orderings of Y, Cb and Cr (listed below).

Component ordering	Color subsampling
Cb,Y,Cr	4:4:4
Y,Cb,Cr	4:2:2
Y,Cr,Cb	4:1:1
Cr,Y,Cb	4:2:0
Cr,Cb,Y	
Cb,Cr,Y	

For completeness and clarity, the layouts that can be expressed with other Properties are listed here, and include the values in other Properties that would express the same layout. If an equivalency exists, the UL should be encoded in the Picture Essence Coding and the equivalent optional Properties shall also be encoded; the latter requirement eases transition with legacy encodings and decoders. If no equivalency exists, the UL shall be encoded, and other appropriate optional Properties should be encoded, but shall not include ReversedByteOrder or PaddingBits.

Note: Legacy MXF files with uncompressed video whose parameters and packing can be unambiguously expressed with the CDCI Picture Descriptor Properties remain valid files.

Layouts can be either interleaved or planar. All are expressed as bit-fields in a byte-stream, otherwise known as big-endian order. Bits within bytes are ordered with most significant bit first. Bit width dimensions are labeled with the words bits, bit, or just b. Layouts cover an entire frame or field, as specified by Frame Layout. Whenever the expression W·H appears in the table below, it refers to Stored Width times Stored Height. Interleaved components appear in adjacent bit-fields in the layout; the simplest of these are 8-bit components which appear as adjacent bytes. Planar layouts are arranged so that all Y components appear together, all Cr components appear together, etc. Most layouts below are interleaved; planar layouts are necessary when the Vertical Subsampling is greater than 1.

Note: In the definitions below, the relative lengths of individual components shown are not to scale.

Item UL (RP 224)	Definition
06.0E.2B.34 04.01.01.0A	
04.01.02.01 01.01.01.01	Interleaved, 4:4:4, 8-bit component. A pattern of Cb, Y, Cr repeated W·H times. Equivalent to: ComponentDepth=8, HorizontalSubsampling=1, VerticalSubsampling=1, ReversedByteOrder=false, PaddingBits=0.
06.0E.2B.34 04.01.01.0A	
04.01.02.01 01.02.01.01	Interleaved, 4:2:2, 8-bit component. A pattern of Cb, Y, Cr, Y repeated (W/2)·H times. This is also known by the 4-cc of '2vuy', and is the format long used by professional video tape formats, transmission protocols, processing equipment (computer and not), and compressions schemes. Equivalent to: ComponentDepth=8, HorizontalSubsampling=2, VerticalSubsampling=1, ReversedByteOrder=false, PaddingBits=0. Stored Width shall be a multiple of 2.
06.0E.2B.34 04.01.01.0A	
04.01.02.01 01.02.01.02	Interleaved, 4:2:2, 8-bit component. A pattern of Y, Cb, Y, Cr repeated (W/2)·H times. This is also known by the 4-cc of 'yuv2', and is a format used by PC-based display cards with Y,Cb,Cr to RGB conversion hardware, which is typically used to accelerate MPEG decompression. Equivalent to: ComponentDepth=8, HorizontalSubsampling=2, VerticalSubsampling=1, ReversedByteOrder=false, PaddingBits=0.

Item UL (RP 224)	Definition
06.0E.2B.34 04.01.01.0A 04.01.02.01 01.02.01.03	<p>Also:</p> <p>Planar, 4:2:2, 8-bit component. The Y plane is a pattern of <math>W \cdot H</math> 8-bit Y components. The Cb and Cr planes follow, in order, using a pattern of <math>(W/2) \cdot H</math> 8-bit components. No equivalency, although ComponentDepth=8, HorizontalSubsampling=2, VerticalSubsampling=1 may be specified.</p>
06.0E.2B.34 04.01.01.0A 04.01.02.01 01.02.02.01	<p>Also:</p> <p>Interleaved, 4:2:2, 10-bit component. A pattern of 10-bit components Cr, Y, Cb, Y with a 2-bit pad inserted at the start of every 32 bits. This results in a rotating pattern of three components for every 32-bits. A full rotation occurs every 6 Pixels, so the full pattern is repeated <math>((W/2) \cdot H)/3</math> times. The value of the Pad bits is unspecified. This layout is sometimes referred to as 'v210'. No equivalency, although ComponentDepth=10, HorizontalSubsampling=2, VerticalSubsampling=1 may be specified. Stored Width shall be a multiple of 48.</p>
06.0E.2B.34 04.01.01.0A 04.01.02.01 01.02.02.02	<p>Also:</p> <p>Planar, 4:2:2, 10-bit component. The Y plane is a pattern of one 2-bit pad followed by three 10-bit Y components. This results in a pattern of three components for every 32-bits, so the full pattern for each component is repeated <math>(W \cdot H)/3</math> times. The Cb and Cr planes follow, in order, using the same pattern, but since the sampling is 4:2:2 the full pattern is repeated <math>((W/2) \cdot H)/3</math> times. The value of the Pad bits is unspecified. No equivalency, although ComponentDepth=10, HorizontalSubsampling=2, VerticalSubsampling=1 may be specified. Stored Width shall be a multiple of 3.</p>
06.0E.2B.34 04.01.01.0A 04.01.02.01 01.02.03.01	<p>Also:</p> <p>Interleaved, 4:2:2, 12-bit component. A pattern of 12-bit components Cb, Y, Cr, Y, repeated <math>(W/2) \cdot H</math> times. Equivalent to: ComponentDepth=12, HorizontalSubsampling=2, VerticalSubsampling=1, ReversedByteOrder=false, PaddingBits=0. Stored Width shall be a multiple of 2.</p>
06.0E.2B.34 04.01.01.0A 04.01.02.01 01.02.04.01	<p>Also:</p> <p>Interleaved, 4:2:2, 16-bit component. A pattern of 16-bit components Cb, Y, Cr, Y, repeated <math>(W/2) \cdot H</math> times. Equivalent to: ComponentDepth=16, HorizontalSubsampling=2, VerticalSubsampling=1, ReversedByteOrder=false, PaddingBits=0. Stored Width shall be a multiple of 2.</p>



Item UL (RP 224)	Definition
06.0E.2B.34	<div><div><div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Y</div><div>Y</div><div>...</div><div>Y</div></div></div><div><div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Cb</div><div>Cb</div><div>...</div><div>Cb</div></div></div><div><div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Cr</div><div>Cr</div><div>...</div><div>Cr</div></div></div></div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Y</div></div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Cb</div></div><div><div><div>← 8 bits →</div><div>← 8 bits →</div><div>← 8 bits →</div></div><div>Cr</div></div></div><div>Also:<div><div><div>← W →</div><div>Y</div><div>↑ H</div><div>↓</div></div><div><div>← W/2 →</div><div>Cb</div><div>↑ H/2</div><div>↓</div></div><div><div>← W/2 →</div><div>Cr</div><div>↑ H/2</div><div>↓</div></div></div></div></div></div>
04.01.01.0A	
04.01.02.01	
01.03.01.02	
	<p>Planar, 4:2:0, 8-bit component; all Y component, followed by all Cb, followed by all Cr. This layout is used in C24 VC-1 reference encoder input, and decoder output. It is sometimes referred to as YV12. No equivalency, although ComponentDepth=8, HorizontalSubsampling=2, VerticalSubsampling=2 may be specified. Stored Width and Height shall be a multiple of 2.</p>

### G.2.26 Component Depth

Component Depth is a Property, whose unsigned 32-bit integer value shall specify the number of active bits per component sample. Typical values are 8 or 10.

The following Distinguished Values of this Property indicate defined floating point component sample formats:

1. 253→HALF (floating point 16-bit value)
2. 254→IEEE floating point 32-bit value
3. 255→IEEE floating point 64-bit value

Component samples of type HALF are stored as 16-bit floating-point numbers. HALF numbers have 1 sign bit, 5 exponent bits, and 10 mantissa bits. The interpretation of the sign, exponent and mantissa is analogous to IEEE-754 floating-point numbers. HALF supports normalized and denormalized numbers, infinities and NaNs (Not A Number). The range of representable numbers is roughly  $6.0 \times 10^{-8}$  -  $6.5 \times 10^4$ ; numbers smaller than  $6.1 \times 10^{-5}$  are denormalized. Conversions from float to half round the mantissa to 10 bits; the 13 least significant bits are lost. Conversions from half to float are lossless; all half numbers are exactly representable as float values.

Note: This data type is the same as that defined for the OpenEXR file format (see Annex I Bibliography).

### G.2.27 Horizontal Subsampling

Horizontal Subsampling is a Property, whose unsigned 32-bit integer value shall specify the horizontal subsampling factor of the color difference samples relative to the luma samples in the Stored Rectangle.

The following values of Horizontal Subsampling and Vertical Subsampling shall be used for the given color difference component sampling ratios.

Color difference component sampling	Horizontal Subsampling	Vertical Subsampling
4:4:4	1	1 (or omitted)
4:2:2	2	1 (or omitted)
4:2:0	2	2
4:1:1	4	1 (or omitted)

### G.2.28 Vertical Subsampling

Vertical Subsampling is a Property, whose unsigned 32-bit integer value shall specify the vertical subsampling factor of the color difference samples relative to the luma samples in the Stored Rectangle.

Typical values are 1 or 2. Other values may be used.

The default value shall be one (1).

### G.2.29 Color Siting

Color Siting is a Property, whose enumerated unsigned 8-bit value shall specify how to compute subsampled color difference values.

The defined values shall be:

00h	coSiting	The first luma sample of the image is co-sited with the first color difference sample(s), as in ITU-R Rec 601, SMPTE ST 314 4:1:1 or MPEG-2 4:2:2.
01h	horizontal mid-point	The color sample is sited at the point horizontally midway between the luma sample on each line.
02h	threeTap	reserved
03h	Quincunx	Color samples are sited at the point midway between two adjacent luma samples on two adjacent lines, as in MPEG-1 4:2:0
04h	Rec601	Color samples are known to be sited in accordance with ITU-R Rec 601, SMPTE ST 274 and SMPTE ST 296
05h	line alternating	The first luma sample of the image is co-sited with the first Cr color difference sample. Cb samples are co-located with Cr samples, but appear on alternating lines, as in IEC 61834-2 625-50 signals.
06h	vertical midpoint	The color sample is sited vertically midway between the luma sample on each column, as in MPEG-2 4:2:0 and H.264/AVC 4:2:0 (default).
FFh	Unknown	The siting of the color samples is unknown

The definitions of 00h (coSiting) and 04h (Rec 601-7) are equivalent. The value of 04h is deprecated. New MXF encoders shall use the value of 00h instead.

The default value shall be FFh.

### G.2.30 PaddingBits

PaddingBits is a Property, whose 16-bit integer shall specify the number of bits to round up each component sample to the stored size of each component sample.

If the Property is not present, MXF applications that require it shall use a value of zero (0).

### G.2.31 Alpha Sample Depth

Alpha Sample Depth is a Property, whose unsigned 32-bit integer value shall specify the number of active bits per sample in the alpha channel. Typical values are 8 or 10.

The default value shall be zero (0).

### G.2.32 Black Ref Level

Black Ref Level is a Property, whose unsigned 32-bit integer value shall specify the luma sample value for reference black level.

Example: For 8-bit ITU-R BT.601-7, the value is 16; for 10-bit ITU-R BT.601-7, the value is 64.

If the Property is not present, MXF applications that require it shall determine the default the value to be used as follows:

1. The Picture Essence Coding Label shall be used to determine the number of bits in which Luminance values are coded.
2. If the Signal Standard Property is present, the default value of Black Ref Level shall be the reference black level defined by that Signal Standard, coded with the number of bits indicated by the Picture Essence Coding Property.
3. If the Signal Standard Property is not present or if it holds its default value, the default value of Black Ref Level shall be: 16 for 8 bit CDCI signals, 64 for 10 bit CDCI signals and zero (0) for all other signals.

### G.2.33 White Ref Level

White Ref Level is a Property, whose unsigned 32-bit integer value shall specify the luma sample value for reference white level.

Example: For ITU-R BT.601, 8-bit video, the value is 235; for ITU-R BT.601-7, 10-bit video, the value is 940. If the Signal Standard Property is present, MXF applications that require it shall determine the default the default value to be used as follows:

1. The Picture Essence Coding Label shall be used to determine the number of bits in which Luminance values are coded.
2. If the Signal Standard Property is present, the default value of White Ref Level shall be the reference white level defined by that Signal Standard, coded with the number of bits indicated by the Picture Essence Coding Property.
3. If the Signal Standard Property is not present or if it holds its default value, the default value of White Ref Level shall be: 235 for 8-bit CDCI signals, 940 for 10-bit CDCI signals, and maximum unsigned integer value for the component size for all other signals.

### G.2.34 Color Range

Color Range is a Property, whose unsigned 32-bit integer value shall specify the number of distinct values allowed for color difference samples. Typical values are 225 or 897 (for 8- or 10-bit samples, respectively).

Example: For ITU-R BT.601 8-bit video, the value is 225. For ITU-R BT.601-7 10-bit video, the value is 897.

If the Signal Standard Property is present, the default value of Color Range shall be the color range value defined by the standard that is indicated by the value of the Signal Standard Property. If the Signal Standard property is not present or if it holds its default value, the default value of Color Range shall be the maximum unsigned integer value for the component size.

### G.2.35 Reversed Byte Order

This Property is deprecated. New designs shall use the Picture Essence Coding Property.

Reversed Byte Order is a Property, whose Boolean value shall be one (TRUE) if the luma sample precedes the color difference sample in the stored data. It shall be zero (FALSE) if the sample order conforms to the ITU-R BT.601-7 standard (first color difference sample precedes luma sample).

Although the title of this Property is “byte order”, it shall refer to the whole sample.

If the Property is not present, MXF applications that require it shall use the Picture Essence Coding Property.

If both the Reversed ByteOrder and PictureEssenceCoding Properties are missing, MXF applications that require Reversed Byte Order shall use a value of zero (FALSE).

This Property is provided for compatibility with some formats which may be wrapped by MXF (for example some uncompressed disk formats). New implementations should use the Picture Essence Coding Property and Reversed Byte Order should be omitted unless backwards compatibility with the existing format requires a value of one (TRUE).

### G.2.36 PixelLayout

PixelLayout is a Property (for RGBA sampling only), whose value shall define the stored format of each RBGA pixel. It is defined in detail in Annex G.2.40.

### G.2.37 Palette

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

Palette is a Property, whose value shall define the restricted list of RGBA Pixel values that is used in the Essence Element. It is defined in detail in Annex G.2.40.

### G.2.38 PaletteLayout

PaletteLayout is a Property, whose value describes the format of each entry in the Palette. It is defined in detail in Annex G.2.40.

### G.2.39 Scanning Direction

Scanning Direction is a Property, whose 8-bit enumerated value shall specify the scanning direction of the image.

Note: It exactly matches the equivalent property in SMPTE ST 268.

If not present, MXF applications that require it shall use a value of zero.

Property Name	Type	Explanation
ScanningDirection	Orientation	Specifies the scanning direction of the image, according to the following enumerated values:
		Code      line direction (followed by)      frame direction
		0          left to right,                      top to bottom
		1          right to left,                     top to bottom
		2          left to right,                     bottom to top
		3          right to left,                     bottom to top
		4          top to bottom,                    left to right
		5          top to bottom,                    right to left
		6          bottom to top,                    left to right
		7          bottom to top,                    right to left
		all other values      reserved for future use

## G.2.40 Pixel Layout

Pixel Layout shall be a property of type RGBALayout, whose value shall specify the type, order and size of the components within the Pixel.

Property Name	Type	Explanation																																																																																							
PixelLayout	RGBALayout	<p>The RGBALayout type shall be a fixed-size 8 element sequence with a total length of 16 bytes, where each element shall consist of the RGBAComponent type with the following fields:</p> <table> <tr> <td>Code</td><td>UInt8</td><td>Enumerated value specifying component 0→RGBALayout terminator</td></tr> <tr> <td>Depth</td><td>UInt8</td><td>Integer specifying the number of bits occupied (see also G.2.26) 1→32 indicates integer depth 253→HALF (floating point 16-bit value) 254→IEEE floating point 32-bit value 255→IEEE floating point 64-bit value 0→RGBALayout terminator</td></tr> </table> <p>For each component in the Pixel, one of the following Codes or the terminator shall be specified (explained below):</p> <table> <tr> <td>Code</td><td>ASCII</td><td>meaning</td></tr> <tr><td>0x52</td><td>'R'</td><td>Red component</td></tr> <tr><td>0x47</td><td>'G'</td><td>Green component</td></tr> <tr><td>0x42</td><td>'B'</td><td>Blue component</td></tr> <tr><td>0x41</td><td>'A'</td><td>Alpha component</td></tr> <tr><td>0x72</td><td>'r'</td><td>Red component (LSBs)</td></tr> <tr><td>0x67</td><td>'g'</td><td>Green component (LSBs)</td></tr> <tr><td>0x62</td><td>'b'</td><td>Blue component (LSBs)</td></tr> <tr><td>0x61</td><td>'a'</td><td>Alpha component (LSBs)</td></tr> <tr><td>0x46</td><td>'F'</td><td>Fill component</td></tr> <tr><td>0x50</td><td>'P'</td><td>Palette code</td></tr> <tr><td>0x55</td><td>'U'</td><td>Color Difference Sample (e.g. U, Cb, I etc.)</td></tr> <tr><td>0x56</td><td>'V'</td><td>Color Difference Sample (e.g. V, Cr, Q etc.)</td></tr> <tr><td>0x57</td><td>'W'</td><td>Composite Video</td></tr> <tr><td>0x58</td><td>'X'</td><td>Non co-sited luma component</td></tr> <tr><td>0x59</td><td>'Y'</td><td>Luma component</td></tr> <tr><td>0x5A</td><td>'Z'</td><td>Depth component (SMPTE ST 268 compatible)</td></tr> <tr><td>0x75</td><td>'u'</td><td>Color Difference Sample (e.g. U, Cb, I etc.) (LSBs)</td></tr> <tr><td>0x76</td><td>'v'</td><td>Color Difference Sample (e.g. V, Cr, Q etc.) (LSBs)</td></tr> <tr><td>0x77</td><td>'w'</td><td>Composite Video (LSBs)</td></tr> <tr><td>0x78</td><td>'x'</td><td>Non co-sited luma component (LSBs)</td></tr> <tr><td>0x79</td><td>'y'</td><td>Luma component (LSBs)</td></tr> <tr><td>0x7A</td><td>'z'</td><td>Depth component (LSBs) (SMPTE ST 268 compatible)</td></tr> <tr><td>0xD8</td><td>'X'</td><td>The DCDM X color component (see SMPTE ST 428-1 X')</td></tr> <tr><td>0xD9</td><td>'Y'</td><td>The DCDM Y color component (see SMPTE ST 428-1 Y')</td></tr> <tr><td>0xDA</td><td>'Z'</td><td>The DCDM Z color component (see SMPTE ST 428-1 Z')</td></tr> <tr><td>0x00</td><td></td><td>Terminates list of components</td></tr> </table> <p>A Fill component indicates unused bits. After the components have been specified, the remaining Code and Size fields shall be set to zero (0).</p>	Code	UInt8	Enumerated value specifying component 0→RGBALayout terminator	Depth	UInt8	Integer specifying the number of bits occupied (see also G.2.26) 1→32 indicates integer depth 253→HALF (floating point 16-bit value) 254→IEEE floating point 32-bit value 255→IEEE floating point 64-bit value 0→RGBALayout terminator	Code	ASCII	meaning	0x52	'R'	Red component	0x47	'G'	Green component	0x42	'B'	Blue component	0x41	'A'	Alpha component	0x72	'r'	Red component (LSBs)	0x67	'g'	Green component (LSBs)	0x62	'b'	Blue component (LSBs)	0x61	'a'	Alpha component (LSBs)	0x46	'F'	Fill component	0x50	'P'	Palette code	0x55	'U'	Color Difference Sample (e.g. U, Cb, I etc.)	0x56	'V'	Color Difference Sample (e.g. V, Cr, Q etc.)	0x57	'W'	Composite Video	0x58	'X'	Non co-sited luma component	0x59	'Y'	Luma component	0x5A	'Z'	Depth component (SMPTE ST 268 compatible)	0x75	'u'	Color Difference Sample (e.g. U, Cb, I etc.) (LSBs)	0x76	'v'	Color Difference Sample (e.g. V, Cr, Q etc.) (LSBs)	0x77	'w'	Composite Video (LSBs)	0x78	'x'	Non co-sited luma component (LSBs)	0x79	'y'	Luma component (LSBs)	0x7A	'z'	Depth component (LSBs) (SMPTE ST 268 compatible)	0xD8	'X'	The DCDM X color component (see SMPTE ST 428-1 X')	0xD9	'Y'	The DCDM Y color component (see SMPTE ST 428-1 Y')	0xDA	'Z'	The DCDM Z color component (see SMPTE ST 428-1 Z')	0x00		Terminates list of components
Code	UInt8	Enumerated value specifying component 0→RGBALayout terminator																																																																																							
Depth	UInt8	Integer specifying the number of bits occupied (see also G.2.26) 1→32 indicates integer depth 253→HALF (floating point 16-bit value) 254→IEEE floating point 32-bit value 255→IEEE floating point 64-bit value 0→RGBALayout terminator																																																																																							
Code	ASCII	meaning																																																																																							
0x52	'R'	Red component																																																																																							
0x47	'G'	Green component																																																																																							
0x42	'B'	Blue component																																																																																							
0x41	'A'	Alpha component																																																																																							
0x72	'r'	Red component (LSBs)																																																																																							
0x67	'g'	Green component (LSBs)																																																																																							
0x62	'b'	Blue component (LSBs)																																																																																							
0x61	'a'	Alpha component (LSBs)																																																																																							
0x46	'F'	Fill component																																																																																							
0x50	'P'	Palette code																																																																																							
0x55	'U'	Color Difference Sample (e.g. U, Cb, I etc.)																																																																																							
0x56	'V'	Color Difference Sample (e.g. V, Cr, Q etc.)																																																																																							
0x57	'W'	Composite Video																																																																																							
0x58	'X'	Non co-sited luma component																																																																																							
0x59	'Y'	Luma component																																																																																							
0x5A	'Z'	Depth component (SMPTE ST 268 compatible)																																																																																							
0x75	'u'	Color Difference Sample (e.g. U, Cb, I etc.) (LSBs)																																																																																							
0x76	'v'	Color Difference Sample (e.g. V, Cr, Q etc.) (LSBs)																																																																																							
0x77	'w'	Composite Video (LSBs)																																																																																							
0x78	'x'	Non co-sited luma component (LSBs)																																																																																							
0x79	'y'	Luma component (LSBs)																																																																																							
0x7A	'z'	Depth component (LSBs) (SMPTE ST 268 compatible)																																																																																							
0xD8	'X'	The DCDM X color component (see SMPTE ST 428-1 X')																																																																																							
0xD9	'Y'	The DCDM Y color component (see SMPTE ST 428-1 Y')																																																																																							
0xDA	'Z'	The DCDM Z color component (see SMPTE ST 428-1 Z')																																																																																							
0x00		Terminates list of components																																																																																							
Palette	DataValue	<p>A list of color values that are used to specify an image.</p> <p>Size specified by parsing the PaletteLayout Property.</p>																																																																																							
PaletteLayout	RGBALayout	<p>A list of PixelLayout elements which specifies the order and size of the color components as they are stored in the palette.</p>																																																																																							

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

An alpha value may be included in each Pixel. The alpha value determines the transparency of the color.

The PixelLayout Property shall specify the order that the color components are stored in the image, the number of bits needed to store a Pixel, and the bits allocated to each component. This covers a wide variety of scanning and packing formats, including all those of SMPTE ST 268.

The 'R', 'G', 'B' and 'A' or 'M' codes shall specify Red, Green, Blue and Alpha or Mask components. The Fill ('F') code allows for insertion of extra bits to Pack the components into convenient word sizes. If the PixelLayout Property includes an 'R', 'G', or 'B', then it shall not include a 'P'.

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

The Palette ('P') code shall specify palletized color sampling, in which each Pixel shall be defined by an index into a Pixel palette. If the PixelLayout Property includes a 'P', then it shall not include an 'R', 'G', or 'B'. If the PixelLayout Property includes a 'P', then the RGBADescriptor Object shall include the Palette and PaletteLayout Properties. The Palette and PaletteLayout Properties specify the color palette itself and the structure used to store each color in the palette.

The 'U', 'V', 'W', 'X', 'Y' and 'Z' codes shall specify abnormal subsampled color and single component sampling (all standardized methods use the CDCI Picture Essence Descriptor). If the PixelLayout Property includes any of these, then it shall not include a 'P'.

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

The X, Y, Z codes shall specify color components for use in Digital Cinema imagery as defined in SMPTE ST 428-1.

RGBA content data may be converted to CDCI and then compressed using a preferred compression scheme. Once the data has been converted and compressed, it shall be described by a CDCI Picture Essence Descriptor (see Annex F.4.2).

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

Example: Component 4:2:2:4

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

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

Example: One of the formats supported by SMPTE ST 268

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

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

Note: MXF Descriptors provide a mechanism to support a similar range of image types to those covered by SMPTE ST 268.

Pixels of type HALF are stored as 16-bit floating-point numbers as described in Annex G.2.26.

## Annex H Static Local Tags Assigned by MXF Specifications (normative)

Column "Local Tag" of the table below contains the complete list of static Local Tags that may be used in MXF Local Sets. MXF Encoders shall not use any static Local Tags other than the ones defined in this table.

When used in an MXF Local Set, the static Local Tags shall be mapped to the UL listed in the "Universal Label" column of the table.

Note 1: According to Section 9.2.2, the range of static Local Tags controlled by MXF specifications is 01.00h to 7F.FFh.

Note 2: The table contains the complete list of static Local Tags assigned by MXF specifications as well as all of those reserved for compatibility with AAF.

**Table H.1 – Static Local Tags**

Local Tag	Universal Label
00.01	06.0E.2B.34.01.01.01.0A.06.01.01.07.16.00.00.00
00.02	06.0E.2B.34.01.01.01.0A.06.01.01.07.17.00.00.00
00.03	06.0E.2B.34.01.01.01.02.06.01.01.07.07.00.00.00
00.04	06.0E.2B.34.01.01.01.02.06.01.01.07.08.00.00.00
00.05	06.0E.2B.34.01.01.01.02.06.01.01.07.13.00.00.00
00.06	06.0E.2B.34.01.01.01.02.03.02.04.01.02.01.00.00
00.07	06.0E.2B.34.01.01.01.02.06.01.01.07.14.01.00.00
00.08	06.0E.2B.34.01.01.01.02.06.01.01.07.01.00.00.00
00.09	06.0E.2B.34.01.01.01.02.06.01.01.07.02.00.00.00
00.0A	06.0E.2B.34.01.01.01.02.06.01.01.07.03.00.00.00
00.0B	06.0E.2B.34.01.01.01.02.06.01.01.07.04.00.00.00
00.0C	06.0E.2B.34.01.01.01.02.03.01.02.02.01.00.00.00
00.0D	06.0E.2B.34.01.01.01.02.06.01.01.07.05.00.00.00
00.0E	06.0E.2B.34.01.01.01.02.06.01.01.07.06.00.00.00
00.0F	06.0E.2B.34.01.01.01.02.03.01.02.03.01.00.00.00
00.10	06.0E.2B.34.01.01.01.02.03.01.02.03.02.00.00.00
00.11	06.0E.2B.34.01.01.01.02.06.01.01.07.09.00.00.00
00.12	06.0E.2B.34.01.01.01.02.06.01.01.07.0A.00.00.00
00.13	06.0E.2B.34.01.01.01.02.03.01.02.03.0B.00.00.00
00.14	06.0E.2B.34.01.01.01.02.06.01.01.07.0B.00.00.00
00.15	06.0E.2B.34.01.01.01.02.03.01.02.03.04.00.00.00
00.16	06.0E.2B.34.01.01.01.02.03.01.02.03.05.00.00.00
00.17	06.0E.2B.34.01.01.01.02.06.01.01.07.0C.00.00.00
00.18	06.0E.2B.34.01.01.01.02.03.01.02.03.03.00.00.00
00.19	06.0E.2B.34.01.01.01.02.06.01.01.07.0D.00.00.00
00.1A	06.0E.2B.34.01.01.01.02.06.01.01.07.0E.00.00.00
00.1B	06.0E.2B.34.01.01.01.02.06.01.01.07.0F.00.00.00
00.1C	06.0E.2B.34.01.01.01.02.06.01.01.07.11.00.00.00



00.1D	06.0E.2B.34.01.01.01.02.03.01.02.03.06.00.00.00
00.1E	06.0E.2B.34.01.01.01.02.06.01.01.07.12.00.00.00
00.1F	06.0E.2B.34.01.01.01.02.03.01.02.03.07.00.00.00
00.20	06.0E.2B.34.01.01.01.02.03.01.02.03.08.00.00.00
00.21	06.0E.2B.34.01.01.01.0A.06.01.01.07.18.00.00.00
00.22	06.0E.2B.34.01.01.01.0A.06.01.01.07.19.00.00.00
01.01	06.0E.2B.34.01.01.01.02.06.01.01.04.01.01.00.00
01.02	06.0E.2B.34.01.01.01.02.05.20.07.01.08.00.00.00
02.01	06.0E.2B.34.01.01.01.02.04.07.01.00.00.00.00.00
02.02	06.0E.2B.34.01.01.01.02.07.02.02.01.01.03.00.00
02.03	06.0E.2B.34.01.01.01.02.03.01.02.10.04.00.00.00
02.04	06.0E.2B.34.01.01.01.07.03.02.01.02.16.00.00.00
02.05	06.0E.2B.34.01.01.01.07.03.01.02.10.08.00.00.00
04.01	06.0E.2B.34.01.01.01.02.01.04.09.01.00.00.00.00
04.02	06.0E.2B.34.01.01.01.02.04.10.01.03.01.09.00.00
04.03	06.0E.2B.34.01.01.01.01.04.10.01.03.01.02.00.00
04.04	06.0E.2B.34.01.01.01.02.01.03.02.01.02.00.00.00
05.01	06.0E.2B.34.01.01.01.02.06.01.01.04.06.01.00.00
05.02	06.0E.2B.34.01.01.01.02.06.01.01.04.02.08.00.00
06.01	06.0E.2B.34.01.01.01.02.07.02.01.03.03.03.00.00
06.02	06.0E.2B.34.01.01.01.02.05.30.04.04.01.00.00.00
08.01	06.0E.2B.34.01.01.01.01.05.30.04.01.00.00.00.00
09.01	06.0E.2B.34.01.01.01.02.06.01.01.04.02.0A.00.00
0B.01	06.0E.2B.34.01.01.01.02.05.30.05.06.00.00.00.00
0B.02	06.0E.2B.34.01.01.01.02.06.01.01.04.06.02.00.00
0B.03	06.0E.2B.34.01.01.01.02.06.01.01.04.06.0A.00.00
0B.04	06.0E.2B.34.01.01.01.02.05.30.05.0C.00.00.00.00
0B.05	06.0E.2B.34.01.01.01.02.06.01.01.04.02.06.00.00
0C.01	06.0E.2B.34.01.01.01.02.06.01.01.04.06.07.00.00
0D.01	06.0E.2B.34.01.01.01.02.06.01.01.04.02.07.00.00
0D.02	06.0E.2B.34.01.01.01.02.05.40.10.01.02.00.00.00
0D.03	06.0E.2B.34.01.01.01.02.05.40.10.01.01.00.00.00
0D.04	06.0E.2B.34.01.01.01.02.05.40.10.01.03.00.00.00
0E.01	06.0E.2B.34.01.01.01.02.06.01.01.03.03.00.00.00
0E.02	06.0E.2B.34.01.01.01.02.06.01.01.03.04.00.00.00
0F.01	06.0E.2B.34.01.01.01.02.06.01.01.04.02.09.00.00
0F.02	06.0E.2B.34.01.01.01.02.06.01.01.04.06.08.00.00
10.01	06.0E.2B.34.01.01.01.02.06.01.01.04.06.09.00.00
11.01	06.0E.2B.34.01.01.01.02.06.01.01.03.01.00.00.00
11.02	06.0E.2B.34.01.01.01.02.06.01.01.03.02.00.00.00
11.03	06.0E.2B.34.01.01.01.07.06.01.01.03.07.00.00.00

11.04	06.0E.2B.34.01.01.01.08.06.01.01.03.08.00.00.00
12.01	06.0E.2B.34.01.01.01.02.07.02.01.03.01.04.00.00
12.02	06.0E.2B.34.01.01.01.02.07.02.02.01.01.05.02.00
12.03	06.0E.2B.34.01.01.01.01.05.30.05.01.00.00.00.00
12.04	06.0E.2B.34.01.01.01.02.07.02.02.01.01.05.03.00
12.05	06.0E.2B.34.01.01.01.01.05.30.05.02.00.00.00.00
14.01	06.0E.2B.34.01.01.01.02.05.30.06.01.01.00.00.00
14.02	06.0E.2B.34.01.01.01.02.05.30.06.02.01.00.00.00
15.01	06.0E.2B.34.01.01.01.02.07.02.01.03.01.05.00.00
15.02	06.0E.2B.34.01.01.01.02.04.04.01.01.02.06.00.00
15.03	06.0E.2B.34.01.01.01.01.04.04.01.01.05.00.00.00
16.01	06.0E.2B.34.01.01.01.02.04.04.01.01.02.01.00.00
16.02	06.0E.2B.34.01.01.01.02.04.07.03.00.00.00.00.00
16.03	06.0E.2B.34.01.01.01.01.04.04.02.01.00.00.00.00
17.01	06.0E.2B.34.01.01.01.01.04.04.01.01.04.00.00.00
18.01	06.0E.2B.34.01.01.01.02.06.01.01.04.02.05.00.00
18.02	06.0E.2B.34.01.01.01.02.07.02.01.03.01.06.00.00
19.01	06.0E.2B.34.01.01.01.02.06.01.01.04.05.01.00.00
19.02	06.0E.2B.34.01.01.01.02.06.01.01.04.05.02.00.00
1A.02	06.0E.2B.34.01.01.01.02.05.30.05.0D.00.00.00.00
1A.03	06.0E.2B.34.01.01.01.02.07.02.01.03.10.02.01.00
1A.04	06.0E.2B.34.01.01.01.02.05.30.05.08.00.00.00.00
1B.01	06.0E.2B.34.01.01.01.02.01.01.15.03.00.00.00.00
1B.02	06.0E.2B.34.01.01.01.02.01.07.01.02.03.01.00.00
1B.03	06.0E.2B.34.01.01.01.02.03.02.03.01.02.01.00.00
1E.01	06.0E.2B.34.01.01.01.02.05.30.05.09.00.00.00.00
1E.02	06.0E.2B.34.01.01.01.01.05.30.05.03.00.00.00.00
1E.03	06.0E.2B.34.01.01.01.02.06.01.01.04.04.01.00.00
1E.06	06.0E.2B.34.01.01.01.02.05.30.05.0A.00.00.00.00
1E.07	06.0E.2B.34.01.01.01.01.05.30.05.04.00.00.00.00
1E.08	06.0E.2B.34.01.01.01.01.05.30.05.05.00.00.00.00
1E.09	06.0E.2B.34.01.01.01.02.06.01.01.04.03.02.00.00
1F.01	06.0E.2B.34.01.01.01.02.06.01.01.04.01.06.00.00
1F.03	06.0E.2B.34.01.01.01.02.05.30.05.0B.01.00.00.00
22.03	06.0E.2B.34.01.01.01.02.05.20.09.01.00.00.00.00
22.04	06.0E.2B.34.01.01.01.02.03.03.03.01.03.00.00.00
22.05	06.0E.2B.34.01.01.01.02.03.03.03.01.02.01.00.00
22.06	06.0E.2B.34.01.01.01.02.01.0A.01.01.01.01.00.00
22.07	06.0E.2B.34.01.01.01.02.06.01.01.04.02.0B.00.00
22.08	06.0E.2B.34.01.01.01.02.01.0A.01.01.03.00.00.00
22.09	06.0E.2B.34.01.01.01.02.05.20.09.02.00.00.00.00

22.0A	06.0E.2B.34.01.01.01.02.05.20.09.03.00.00.00.00
22.0B	06.0E.2B.34.01.01.01.02.05.20.09.04.00.00.00.00
22.0C	06.0E.2B.34.01.01.01.02.05.20.09.05.00.00.00.00
22.0D	06.0E.2B.34.01.01.01.02.05.20.09.06.00.00.00.00
22.0E	06.0E.2B.34.01.01.01.02.05.20.09.07.00.00.00.00
22.0F	06.0E.2B.34.01.01.01.02.05.20.09.08.00.00.00.00
22.10	06.0E.2B.34.01.01.01.02.05.20.09.09.00.00.00.00
22.11	06.0E.2B.34.01.01.01.02.05.20.09.0A.00.00.00.00
22.12	06.0E.2B.34.01.01.01.02.05.20.09.0B.00.00.00.00
22.13	06.0E.2B.34.01.01.01.02.05.20.09.0C.00.00.00.00
22.14	06.0E.2B.34.01.01.01.02.05.20.09.0D.00.00.00.00
22.15	06.0E.2B.34.01.01.01.02.05.20.09.0E.00.00.00.00
22.16	06.0E.2B.34.01.01.01.02.05.20.09.0F.00.00.00.00
23.01	06.0E.2B.34.01.01.01.02.06.01.01.04.01.07.00.00
23.02	06.0E.2B.34.01.01.01.02.06.01.01.04.03.01.00.00
24.01	06.0E.2B.34.01.01.01.01.03.01.02.01.03.00.00.00
26.03	06.0E.2B.34.01.01.01.02.06.01.01.04.05.03.00.00
26.04	06.0E.2B.34.01.01.01.02.06.01.01.04.05.04.00.00
26.05	06.0E.2B.34.01.01.01.02.06.01.01.04.05.05.00.00
26.06	06.0E.2B.34.01.01.01.02.06.01.01.04.05.06.00.00
26.07	06.0E.2B.34.01.01.01.02.06.01.01.04.05.07.00.00
26.08	06.0E.2B.34.01.01.01.02.06.01.01.04.05.08.00.00
26.09	06.0E.2B.34.01.01.01.02.06.01.01.04.05.09.00.00
26.0A	06.0E.2B.34.01.01.01.07.06.01.01.04.05.0A.00.00
26.0B	06.0E.2B.34.01.01.01.07.06.01.01.04.05.0B.00.00
27.01	06.0E.2B.34.01.01.01.02.06.01.01.06.01.00.00.00
27.02	06.0E.2B.34.01.01.01.02.04.07.02.00.00.00.00.00
2B.01	06.0E.2B.34.01.01.01.02.06.01.01.02.01.00.00.00
2F.01	06.0E.2B.34.01.01.01.02.06.01.01.04.06.03.00.00
30.01	06.0E.2B.34.01.01.01.01.04.06.01.01.00.00.00.00
30.02	06.0E.2B.34.01.01.01.01.04.06.01.02.00.00.00.00
30.04	06.0E.2B.34.01.01.01.02.06.01.01.04.01.02.00.00
30.05	06.0E.2B.34.01.01.01.02.06.01.01.04.01.03.00.00
30.06	06.0E.2B.34.01.01.01.05.06.01.01.03.05.00.00.00
31.01	06.0E.2B.34.01.01.01.02.03.03.03.02.02.00.00.00
32.01	06.0E.2B.34.01.01.01.02.04.01.06.01.00.00.00.00
32.02	06.0E.2B.34.01.01.01.01.04.01.05.02.01.00.00.00
32.03	06.0E.2B.34.01.01.01.01.04.01.05.02.02.00.00.00
32.04	06.0E.2B.34.01.01.01.01.04.01.05.01.07.00.00.00
32.05	06.0E.2B.34.01.01.01.01.04.01.05.01.08.00.00.00
32.06	06.0E.2B.34.01.01.01.01.04.01.05.01.09.00.00.00

32.07	06.0E.2B.34.01.01.01.01.04.01.05.01.0A.00.00.00
32.08	06.0E.2B.34.01.01.01.01.04.01.05.01.0B.00.00.00
32.09	06.0E.2B.34.01.01.01.01.04.01.05.01.0C.00.00.00
32.0A	06.0E.2B.34.01.01.01.01.04.01.05.01.0D.00.00.00
32.0B	06.0E.2B.34.01.01.01.01.04.01.05.01.0E.00.00.00
32.0C	06.0E.2B.34.01.01.01.01.04.01.03.01.04.00.00.00
32.0D	06.0E.2B.34.01.01.01.02.04.01.03.02.05.00.00.00
32.0E	06.0E.2B.34.01.01.01.01.04.01.01.01.01.00.00.00
32.0F	06.0E.2B.34.01.01.01.02.05.20.01.02.00.00.00.00
32.10	06.0E.2B.34.01.01.01.02.04.01.02.01.01.01.02.00
32.11	06.0E.2B.34.01.01.01.02.04.18.01.01.00.00.00.00
32.12	06.0E.2B.34.01.01.01.02.04.01.03.01.06.00.00.00
32.13	06.0E.2B.34.01.01.01.02.04.18.01.02.00.00.00.00
32.14	06.0E.2B.34.01.01.01.02.04.18.01.03.00.00.00.00
32.15	06.0E.2B.34.01.01.01.05.04.05.01.13.00.00.00.00
32.16	06.0E.2B.34.01.01.01.05.04.01.03.02.08.00.00.00
32.17	06.0E.2B.34.01.01.01.05.04.01.03.02.07.00.00.00
32.18	06.0E.2B.34.01.01.01.05.04.01.03.02.09.00.00.00
32.19	06.0E.2B.34.01.01.01.09.04.01.02.01.01.06.01.00
32.1A	06.0E.2B.34.01.01.01.02.04.01.02.01.01.03.01.00
33.01	06.0E.2B.34.01.01.01.02.04.01.05.03.0A.00.00.00
33.02	06.0E.2B.34.01.01.01.01.04.01.05.01.05.00.00.00
33.03	06.0E.2B.34.01.01.01.01.04.01.05.01.06.00.00.00
33.04	06.0E.2B.34.01.01.01.01.04.01.05.03.03.00.00.00
33.05	06.0E.2B.34.01.01.01.01.04.01.05.03.04.00.00.00
33.06	06.0E.2B.34.01.01.01.02.04.01.05.03.05.00.00.00
33.07	06.0E.2B.34.01.01.01.02.04.18.01.04.00.00.00.00
33.08	06.0E.2B.34.01.01.01.02.04.01.05.01.10.00.00.00
33.09	06.0E.2B.34.01.01.01.02.04.01.05.03.07.00.00.00
33.0B	06.0E.2B.34.01.01.01.05.03.01.02.01.0A.00.00.00
34.01	06.0E.2B.34.01.01.01.02.04.01.05.03.06.00.00.00
34.03	06.0E.2B.34.01.01.01.02.04.01.05.03.08.00.00.00
34.04	06.0E.2B.34.01.01.01.02.04.01.05.03.09.00.00.00
34.05	06.0E.2B.34.01.01.01.05.04.01.04.04.01.00.00.00
34.06	06.0E.2B.34.01.01.01.05.04.01.05.03.0B.00.00.00
34.07	06.0E.2B.34.01.01.01.05.04.01.05.03.0C.00.00.00
34.08	06.0E.2B.34.01.01.01.05.04.01.05.03.0D.00.00.00
34.09	06.0E.2B.34.01.01.01.05.04.01.05.03.0E.00.00.00
35.01	06.0E.2B.34.01.01.01.05.04.04.01.02.01.00.00.00
35.02	06.0E.2B.34.01.01.01.05.04.04.01.02.02.00.00.00
35.03	06.0E.2B.34.01.01.01.05.04.04.01.02.03.00.00.00

35.04	06.0E.2B.34.01.01.01.05.04.04.01.02.04.00.00.00
37.01	06.0E.2B.34.01.01.01.02.05.02.01.03.01.01.00.00
37.02	06.0E.2B.34.01.01.01.01.06.08.02.01.00.00.00.00
37.03	06.0E.2B.34.01.01.01.01.04.01.03.02.03.00.00.00
37.04	06.0E.2B.34.01.01.01.01.04.01.03.02.04.00.00.00
37.05	06.0E.2B.34.01.01.01.02.05.02.01.03.01.02.00.00
37.06	06.0E.2B.34.01.01.01.02.03.03.03.02.03.00.00.00
38.01	06.0E.2B.34.01.01.01.02.03.03.03.02.01.00.00.00
39.01	06.0E.2B.34.01.01.01.02.04.10.01.03.01.08.00.00
39.02	06.0E.2B.34.01.01.01.02.04.01.08.02.03.00.00.00
39.03	06.0E.2B.34.01.01.01.02.04.10.01.03.01.03.00.00
39.04	06.0E.2B.34.01.01.01.02.04.10.01.03.02.03.00.00
39.05	06.0E.2B.34.01.01.01.02.04.10.01.03.01.06.01.00
39.06	06.0E.2B.34.01.01.01.02.04.10.01.03.01.05.01.00
39.07	06.0E.2B.34.01.01.01.02.04.10.01.03.01.04.01.00
39.08	06.0E.2B.34.01.01.01.02.04.10.01.03.01.07.01.00
3A.01	06.0E.2B.34.01.01.01.02.04.10.01.01.01.01.00.00
3A.02	06.0E.2B.34.01.01.01.02.04.01.04.01.01.00.00.00
3A.03	06.0E.2B.34.01.01.01.02.0D.01.01.01.01.01.01.00
3A.04	06.0E.2B.34.01.01.01.02.04.10.01.01.03.00.00.00
3A.05	06.0E.2B.34.01.01.01.02.04.10.01.01.04.01.00.00
3A.06	06.0E.2B.34.01.01.01.02.04.10.01.01.02.01.00.00
3A.07	06.0E.2B.34.01.01.01.02.04.10.01.01.06.01.00.00
3A.08	06.0E.2B.34.01.01.01.02.04.10.01.01.05.01.00.00
3B.01	06.0E.2B.34.01.01.01.01.03.01.02.01.02.00.00.00
3B.02	06.0E.2B.34.01.01.01.02.07.02.01.10.02.04.00.00
3B.03	06.0E.2B.34.01.01.01.02.06.01.01.04.02.01.00.00
3B.04	06.0E.2B.34.01.01.01.02.06.01.01.04.02.02.00.00
3B.05	06.0E.2B.34.01.01.01.02.03.01.02.01.05.00.00.00
3B.06	06.0E.2B.34.01.01.01.02.06.01.01.04.06.04.00.00
3B.07	06.0E.2B.34.01.01.01.02.03.01.02.01.04.00.00.00
3B.08	06.0E.2B.34.01.01.01.04.06.01.01.04.01.08.00.00
3B.09	06.0E.2B.34.01.01.01.05.01.02.02.03.00.00.00.00
3B.0A	06.0E.2B.34.01.01.01.05.01.02.02.10.02.01.00.00
3B.0B	06.0E.2B.34.01.01.01.05.01.02.02.10.02.02.00.00
3C.01	06.0E.2B.34.01.01.01.02.05.20.07.01.02.01.00.00
3C.02	06.0E.2B.34.01.01.01.02.05.20.07.01.03.01.00.00
3C.03	06.0E.2B.34.01.01.01.02.05.20.07.01.04.00.00.00
3C.04	06.0E.2B.34.01.01.01.02.05.20.07.01.05.01.00.00
3C.05	06.0E.2B.34.01.01.01.02.05.20.07.01.07.00.00.00
3C.06	06.0E.2B.34.01.01.01.02.07.02.01.10.02.03.00.00

3C.07	06.0E.2B.34.01.01.01.02.05.20.07.01.0A.00.00.00
3C.08	06.0E.2B.34.01.01.01.02.05.20.07.01.06.01.00.00
3C.09	06.0E.2B.34.01.01.01.02.05.20.07.01.01.00.00.00
3C.0A	06.0E.2B.34.01.01.01.01.01.01.15.02.00.00.00.00
3D.01	06.0E.2B.34.01.01.01.04.04.02.03.03.04.00.00.00
3D.02	06.0E.2B.34.01.01.01.04.04.02.03.01.04.00.00.00
3D.03	06.0E.2B.34.01.01.01.05.04.02.03.01.01.01.00.00
3D.04	06.0E.2B.34.01.01.01.01.04.02.01.01.03.00.00.00
3D.05	06.0E.2B.34.01.01.01.01.04.02.01.01.01.00.00.00
3D.06	06.0E.2B.34.01.01.01.02.04.02.04.02.00.00.00.00
3D.07	06.0E.2B.34.01.01.01.05.04.02.01.01.04.00.00.00
3D.08	06.0E.2B.34.01.01.01.05.04.02.05.01.01.00.00.00
3D.09	06.0E.2B.34.01.01.01.05.04.02.03.03.05.00.00.00
3D.0A	06.0E.2B.34.01.01.01.05.04.02.03.02.01.00.00.00
3D.0B	06.0E.2B.34.01.01.01.05.04.02.03.02.02.00.00.00
3D.0C	06.0E.2B.34.01.01.01.05.04.02.07.01.00.00.00.00
3D.0D	06.0E.2B.34.01.01.01.05.04.02.05.01.06.00.00.00
3D.0F	06.0E.2B.34.01.01.01.05.04.02.03.02.03.00.00.00
3D.10	06.0E.2B.34.01.01.01.05.04.02.05.01.02.00.00.00
3D.11	06.0E.2B.34.01.01.01.05.04.02.05.01.03.00.00.00
3D.12	06.0E.2B.34.01.01.01.05.04.02.05.01.04.00.00.00
3D.13	06.0E.2B.34.01.01.01.05.04.02.05.01.05.00.00.00
3D.15	06.0E.2B.34.01.01.01.05.04.02.03.02.05.00.00.00
3D.16	06.0E.2B.34.01.01.01.05.04.02.03.02.06.00.00.00
3D.21	06.0E.2B.34.01.01.01.05.04.02.05.02.01.01.00.00
3D.22	06.0E.2B.34.01.01.01.05.04.02.05.02.02.01.00.00
3D.23	06.0E.2B.34.01.01.01.05.04.02.05.02.03.01.00.00
3D.24	06.0E.2B.34.01.01.01.05.04.02.05.02.04.01.00.00
3D.25	06.0E.2B.34.01.01.01.05.04.02.05.02.05.01.00.00
3D.26	06.0E.2B.34.01.01.01.05.04.02.05.02.06.01.00.00
3D.27	06.0E.2B.34.01.01.01.05.04.02.05.02.07.01.00.00
3D.28	06.0E.2B.34.01.01.01.05.04.02.05.02.08.01.00.00
3D.29	06.0E.2B.34.01.01.01.08.04.02.03.01.06.00.00.00
3D.2A	06.0E.2B.34.01.01.01.08.04.02.03.01.07.00.00.00
3D.2B	06.0E.2B.34.01.01.01.08.04.02.03.01.08.00.00.00
3D.2C	06.0E.2B.34.01.01.01.08.04.02.03.01.09.00.00.00
3D.2D	06.0E.2B.34.01.01.01.08.04.02.03.01.0A.00.00.00
3D.2E	06.0E.2B.34.01.01.01.08.04.02.03.01.0B.00.00.00
3D.2F	06.0E.2B.34.01.01.01.08.04.02.03.01.0C.00.00.00
3D.30	06.0E.2B.34.01.01.01.08.04.02.03.01.0D.00.00.00
3D.31	06.0E.2B.34.01.01.01.08.04.02.03.01.0E.00.00.00

3D.32	06.0E.2B.34.01.01.01.07.04.02.01.01.05.00.00.00
3D.33	06.0E.2B.34.01.01.01.08.06.01.01.04.06.0F.00.00
3E.01	06.0E.2B.34.01.01.01.03.04.03.03.02.00.00.00.00
3F.01	06.0E.2B.34.01.01.01.04.06.01.01.04.06.0B.00.00
3F.05	06.0E.2B.34.01.01.01.04.04.06.02.01.00.00.00.00
3F.06	06.0E.2B.34.01.01.01.04.01.03.04.05.00.00.00.00
3F.07	06.0E.2B.34.01.01.01.04.01.03.04.04.00.00.00.00
3F.08	06.0E.2B.34.01.01.01.04.04.04.04.01.01.00.00.00
3F.09	06.0E.2B.34.01.01.01.05.04.04.04.01.06.00.00.00
3F.0A	06.0E.2B.34.01.01.01.05.04.04.04.02.05.00.00.00
3F.0B	06.0E.2B.34.01.01.01.05.05.30.04.06.00.00.00.00
3F.0C	06.0E.2B.34.01.01.01.05.07.02.01.03.01.0A.00.00
3F.0D	06.0E.2B.34.01.01.01.05.07.02.02.01.01.02.00.00
3F.0E	06.0E.2B.34.01.01.01.05.04.04.04.01.07.00.00.00
3F.0F	06.0E.2B.34.01.01.01.0A.04.06.02.04.00.00.00.00
3F.10	06.0E.2B.34.01.01.01.0A.04.06.02.05.00.00.00.00
3F.11	06.0E.2B.34.01.01.01.0E.04.04.05.01.00.00.00.00
3F.12	06.0E.2B.34.01.01.01.0E.04.06.02.06.00.00.00.00
3F.13	06.0E.2B.34.01.01.01.0E.04.04.05.02.00.00.00.00
40.01	06.0E.2B.34.01.01.01.01.01.02.01.01.01.00.00.00
41.01	06.0E.2B.34.01.01.01.02.01.04.01.02.01.00.00.00
44.01	06.0E.2B.34.01.01.01.01.01.01.01.15.10.00.00.00.00
44.02	06.0E.2B.34.01.01.01.01.01.03.03.02.01.00.00.00
44.03	06.0E.2B.34.01.01.01.02.06.01.01.04.06.05.00.00
44.04	06.0E.2B.34.01.01.01.02.07.02.01.10.02.05.00.00
44.05	06.0E.2B.34.01.01.01.02.07.02.01.10.01.03.00.00
44.06	06.0E.2B.34.01.01.01.02.03.02.01.02.0C.00.00.00
44.07	06.0E.2B.34.01.01.01.02.03.01.02.10.03.00.00.00
44.08	06.0E.2B.34.01.01.01.07.05.01.01.08.00.00.00.00
44.09	06.0E.2B.34.01.01.01.07.03.01.02.10.07.00.00.00
45.01	06.0E.2B.34.01.01.01.02.07.02.02.01.01.05.01.00
45.02	06.0E.2B.34.01.01.01.01.05.30.02.01.00.00.00.00
45.03	06.0E.2B.34.01.01.01.02.05.30.04.03.00.00.00.00
45.04	06.0E.2B.34.01.01.01.08.06.01.01.04.01.0A.00.00
47.01	06.0E.2B.34.01.01.01.02.06.01.01.04.02.03.00.00
48.01	06.0E.2B.34.01.01.01.02.01.07.01.01.00.00.00.00
48.02	06.0E.2B.34.01.01.01.02.01.07.01.02.01.00.00.00
48.03	06.0E.2B.34.01.01.01.02.06.01.01.04.02.04.00.00
48.04	06.0E.2B.34.01.01.01.02.01.04.01.03.00.00.00.00
49.01	06.0E.2B.34.01.01.01.02.05.30.04.02.00.00.00.00
49.02	06.0E.2B.34.01.01.01.05.07.02.01.03.01.0B.00.00

4B.01	06.0E.2B.34.01.01.01.02.05.30.04.05.00.00.00.00
4B.02	06.0E.2B.34.01.01.01.02.07.02.01.03.01.03.00.00
4B.03	06.0E.2B.34.01.01.01.07.07.02.01.03.01.0C.00.00
4B.04	06.0E.2B.34.01.01.01.07.07.02.01.03.02.03.00.00
4B.05	06.0E.2B.34.01.01.01.07.07.02.01.03.01.0D.00.00
4C.01	06.0E.2B.34.01.01.01.02.06.01.01.04.01.04.00.00
4C.11	06.0E.2B.34.01.01.01.07.06.01.01.04.03.05.00.00
4D.01	06.0E.2B.34.01.01.01.02.05.30.05.07.00.00.00.00
4D.11	06.0E.2B.34.01.01.01.07.06.01.01.04.03.04.00.00
4D.12	06.0E.2B.34.01.01.01.07.06.01.01.04.01.09.00.00
4E.01	06.0E.2B.34.01.01.01.02.06.01.01.04.01.05.00.00
4E.02	06.0E.2B.34.01.01.01.02.06.01.01.04.06.06.00.00
4E.11	06.0E.2B.34.01.01.01.07.04.09.02.01.00.00.00.00
4E.12	06.0E.2B.34.01.01.01.08.04.09.03.00.00.00.00.00
4F.01	06.0E.2B.34.01.01.01.08.04.06.08.02.00.00.00.00
4F.02	06.0E.2B.34.01.01.01.08.04.06.09.03.00.00.00.00
4F.03	06.0E.2B.34.01.01.01.08.04.07.04.00.00.00.00.00
50.01	06.0E.2B.34.01.01.01.02.03.02.01.02.09.01.00.00
50.03	06.0E.2B.34.01.01.01.02.03.02.01.02.0A.01.00.00
51.01	06.0E.2B.34.01.01.01.02.03.01.02.10.02.00.00.00
52.12	06.0E.2B.34.01.01.01.08.04.09.04.01.00.00.00.00
54.01	06.0E.2B.34.01.01.01.08.01.02.01.04.01.00.00.00
54.02	06.0E.2B.34.01.01.01.08.01.02.01.06.01.00.00.00
54.03	06.0E.2B.34.01.01.01.08.01.03.06.06.01.00.00.00
55.01	06.0E.2B.34.01.01.01.08.01.03.06.04.01.00.00.00
56.01	06.0E.2B.34.01.01.01.09.01.03.04.06.00.00.00.00
56.02	06.0E.2B.34.01.01.01.09.01.03.04.07.00.00.00.00
57.01	06.0E.2B.34.01.01.01.09.07.02.05.01.00.00.00.00
57.02	06.0E.2B.34.01.01.01.09.07.02.05.03.00.00.00.00
57.03	06.0E.2B.34.01.01.01.09.07.02.05.02.00.00.00.00
58.01	06.0E.2B.34.01.01.01.09.06.01.01.03.09.00.00.00
58.02	06.0E.2B.34.01.01.01.09.06.01.01.03.0A.00.00.00
58.03	06.0E.2B.34.01.01.01.09.06.01.01.03.0B.00.00.00
58.04	06.0E.2B.34.01.01.01.09.06.01.01.03.0C.00.00.00
61.01	06.0E.2B.34.01.01.01.05.06.01.01.04.02.0C.00.00
61.02	06.0E.2B.34.01.01.01.04.01.07.01.05.00.00.00.00
61.03	06.0E.2B.34.01.01.01.05.01.07.01.06.00.00.00.00

The entry in the Local Tag column identifies the static Local Tag value.

The entry in the Universal Label column identifies the Universal Label to which the static Local Tag is associated.



## Bibliography

Advanced Authoring Format, <https://www.amwa.tv/aaf>

AES3-2009, AES Standard for Digital Audio Engineering — Serial Transmission Format for Two-Channel Linearly Represented Digital Audio Data

AES31-2:2019, AES Standard on Network and File Transfer of Audio — Audio-File Transfer and Exchange — File Format for Transferring Digital Audio Data Between Systems of Different Type and Manufacturer

ATSC A/52:2018, <http://www.atsc.org>, A/52 Specification — Equivalent of Normative: ITU-R Recommendation BS.1196-7 (2019) (Annexes 2, 3): Audio Coding for Digital Terrestrial Television Broadcasting

EBU / SMPTE Task Force for Harmonized Standards for the Exchange of Program Material as Bitstreams — 1998, <http://www.smpste.org> and <http://www.ebu.ch>

EBU Tech 3285-2011, BWF — A Format for Audio Data Files in Broadcasting

ETSI TR 101 154 (2000-07), Implementation Guidelines for the Use of MPEG-2 Systems, Video and Audio in Satellite, Cable and Terrestrial Broadcasting Applications (for Definition of AFD)

IETF RFC 2781 (02/00), UTF-16 — An Encoding of ISO 10646

ISO/IEC 646:1991, Information Technology — ISO 7-Bit Coded Character Set for Information Interchange

ISO/IEC 11578:1996, Information Technology — Open Systems Interconnection — Remote Procedure Call, (RPC) Annex A, Universally Unique Identifier

ISO/IEC 13818-2:2013, Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Video

ITU-R BT.470-7 (02/05), Conventional Analogue Television Systems

ITU-R BT.601-7 (03/11), Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide Screen 16:9 Aspect Ratios

ITU-R BT.656-5 (12/07), Interface for Digital Component Video Signals in 525-Line and 625-Line Television Systems Operating at the 4:2:2 Level of Recommendation ITU-R BT.601

OpenEXR File Format, <http://www.openexr.org>

SMPTE ST 12-1:2014, Television — Time and Control Code

SMPTE ST 125:2013 Television — Component Video Signal 4:2:2 — Bit-Parallel Digital Interface

SMPTE ST 268-1:2014, File Format for Digital Moving-Picture Exchange (DPX), Version 2.0

SMPTE ST 268-2:2018, Digital Moving-Picture Exchange (DPX) – Format Extensions for High Dynamic Range and Wide Color Gamut

SMPTE ST 274:2008, Television — 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates

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

SMPTE ST 296:2012 Television — 1280 x 720 Progressive Image Sample Structure — Analog and Digital Representation and Analog Interface

SMPTE ST 314:2005, Television — Data Structure for DV-Based Audio, Data and Compressed Video — 25 and 50 Mb/s

SMPTE ST 331:2011, Television — Element and Metadata Definitions for the SDTI-CP

SMPTE ST 352:2013, Payload Identification Codes for Serial Digital Interfaces

SMPTE ST 378:2004, Television — Material Exchange Format (MXF) — Operational Pattern 1a (Single Item, Single Package)

SMPTE ST 379-1:2009, Television — Material Exchange Format (MXF) — MXF Generic Container

SMPTE ST 379-2:2010, Television — Material Exchange Format (MXF) — MXF Constrained Generic Container

SMPTE ST 380:2004, Television — Material Exchange Format (MXF) — Descriptive Metadata Scheme-1

SMPTE ST 381-1-2005, Television — Material Exchange Format (MXF) — Mapping MPEG Streams into the MXF Generic Container

SMPTE ST 382:2007, Material Exchange Format (MXF) — Mapping AES3 and Broadcast Wave Audio into the MXF Generic Container

SMPTE ST 383:2008, Television — Material Exchange Format (MXF) — Mapping DV-DIF Data to the MXF Generic Container

SMPTE ST 384:2005, Television — Material Exchange Format (MXF) — Mapping of Uncompressed Pictures into the Generic Container

SMPTE ST 385:2012, Television — Material Exchange Format (MXF) — Mapping SDTI-CP Essence and Metadata into the MXF Generic Container

SMPTE ST 386:2004, Television — Material Exchange Format (MXF) — Mapping Type D-10 Essence Data to the MXF Generic Container

SMPTE ST 387:2004, Television — Material Exchange Format (MXF) — Mapping Type D-11 Essence Data to the MXF Generic Container

SMPTE ST 388:2004, Television — Material Exchange Format (MXF) — Mapping A-law Coded Audio into the MXF Generic Container

SMPTE ST 389:2005, Television — Material Exchange Format (MXF) — MXF Generic Container Reverse Play System Element

SMPTE ST 390:2011, Material Exchange Format (MXF) — Specialized Operational Pattern “OP-Atom” (Simplified Representation of a Single Item)

SMPTE ST 391:2004, Television — Material Exchange Format (MXF) — Operational Pattern 1b (Single Item, Ganged Packages)

SMPTE ST 392:2013, Television — Material Exchange Format (MXF) — Operational Pattern 2a (Play-List Items, Single Package)

SMPTE ST 393:2004, Television — Material Exchange Format (MXF) — Operational Pattern 2b (Play-List Items, Ganged Packages)

SMPTE ST 394:2006, Television — Material Exchange Format (MXF) — System Scheme 1 for the MXF Generic Container

SMPTE ST 405:2006, Television — Material Exchange Format (MXF) — Elements and Individual Data Items for the MXF Generic Container System Scheme 1

SMPTE ST 407:2006, Television — Material Exchange Format (MXF) — Operational Patterns 3a and 3b

SMPTE ST 408:2006, Television — Material Exchange Format (MXF) — Operational Patterns 1c, 2c and 3c

SMPTE ST 410:2008, Material Exchange Format — Generic Stream Partition

SMPTE ST 422:2019, Material Exchange Format — Mapping JPEG 2000 Codestreams into the MXF Generic Container

SMPTE ST 428-1:2019, D-Cinema Distribution Master (DCDM) — Image Characteristics

SMPTE ST 429-4:2006, D-Cinema Packaging — MXF JPEG 2000 Application

SMPTE ST 429-6:2006, D-Cinema Packaging — MXF Track File Essence Encryption

SMPTE ST 434:2014, Material Exchange Format — XML Encoding for Metadata and File Structure Information

SMPTE ST 436-1:2013, Television — MXF Mappings for VBI Lines and Ancillary Data Packets

SMPTE ST 2019-4:2016, Mapping VC-3 Coding Units into the MXF Generic Container

SMPTE ST 2037:2009, Mapping VC-1 into the MXF Generic Container

SMPTE RP 210:2007, Metadata Dictionary Registry of Metadata Elements Descriptions (superseded by SMPTE Metadata Registers, <https://registry.smpte-ra.org/>)

SMPTE RP 224, SMPTE Labels Register

SMPTE ST 381-3:2017, Material Exchange Format — Mapping AVC Streams into the MXF Generic Container

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

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