EG 42-2004

SMPTE ENGINEERING GUIDELINE

for Television — Material Exchange Format (MXF) — MXF Descriptive Metadata

Table of contents

1 Scope

The MXF standard (SMPTE 377M) provides for descriptive metadata schemes as “plug-ins” to the MXF header metadata. This guideline is intended to provide guidance for the use of MXF descriptive metadata schemes. This guideline is a supplement to SMPTE EG 41.

This guideline explains the common structural metadata components by which all MXF descriptive metadata schemes can be related to the essence they describe. Additional constraints required for MXF descriptive metadata schemes are also described.

This guideline describes how MXF descriptive metadata is related to the essence using the structural metadata defined in SMPTE 377M. It also describes how MXF descriptive metadata is coded as KLV data for use in MXF files and as XML for text-based metadata exchange. This guideline further provides guidelines for implementing MXF descriptive metadata schemes.

Descriptive metadata is a complex topic and requires at least some basic understandings of the various attributes of descriptive metadata together with data modeling techniques. This guideline provides annexes that summarize these topics.

Approved September 22, 2004

2 Normative references

The following documents contain provisions which, through reference in this text, constitute provisions of this guideline. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this guideline are encouraged to investigate the possibility of applying the most recent edition of the documents indicated below.

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

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

SMPTE 377M-2004, Television — Material Exchange Format (MXF) — File Format Specification (Standard)

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

3 Glossary of acronyms, terms and data types

The full glossary of acronyms, terms and data types used in the MXF specification is given in the MXF file format specification. It is not repeated here to avoid any divergence of meaning.

3.1 Acronyms

AAF: Advanced Authoring Format (www.aafassociation.org)
DM: Descriptive Metadata
DMS: Descriptive Metadata Scheme
HTML: Hyper-text Markup Language
ISO: International Standards Organization
IP: Intellectual Property (as used in rights management)
O-O: Object Oriented
TFHS: Task Force for the Harmonization of Standards (Bibliography)
UML: Unified Modeling Language
XML: eXtensible Markup Language (text-based data coding)

3.2 Terms

Element: An atomic constituent of a data model.
Property: A named value denoting the characteristics of an element.
Attribute: A named property of a classifier that describes a range of values that instances of a property may hold.
Entity: An entity is an abstract term that can mean a single data element, a set of data elements or a higher construct.

3.3 Data types

None.

4 Introduction

This guideline is intended to provide guidance for implementing descriptive metadata within the header metadata of an MXF file (SMPTE 377M). This guideline supplements the MXF Engineering Guideline (SMPTE EG 41).

This guideline is divided into a number of core sections, each section describing a particular aspect of descriptive metadata. The sections (together with their section number) are as follows:

1. Section 4: This introduction, including the usage of descriptive metadata and a catalogue of user requirements for descriptive metadata.

2. Section 5: The mechanisms by which descriptive metadata in the MXF header metadata may be linked to the essence in the MXF essence container.

3. Section 6: A description of the issues related to implementing a descriptive metadata model from the perspective of both MXF and AAF.

4. Section 7: How to implement descriptive metadata as a sequence of KLV packets in the MXF header metadata and as XML for text-based interfacing.

5. Section 8: Guidance for implementing both MXF descriptive metadata and non-MXF descriptive metadata.

6. Annex A: Provides a discussion of metadata characteristics.

7. Annex B: Provides an introduction to data modelling techniques.

8. Annex C: Provides an example of the options for XML coding.

9. Annex D: Bibliography of useful reading.

4.1 Dimensions of descriptive metadata use

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

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

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

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

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

5. It may have different levels of stability with elements having durable values that remain stable or transient values that may frequently change.

In the header metadata of an MXF file, the metadata is persisted and may be distributed over many copies. Thus descriptive metadata in an MXF file should include only that metadata which can be considered copy persistent. This does not mean that a value can never change between copies, but that, once written, individual copies must be edited where changes are required. For some types of metadata, this is not only acceptable, but encouraged, e.g. a property that defines the number of copies made.

One of the primary aims of this document is to identify that metadata which can be used to enhance the description of the content in a file. Metadata that is appropriate for use in databases is not part of this guideline. Annex A describes metadata characteristics that may aid the reader to understand the attributes of different kinds of metadata items.

4.2 User requirements for descriptive metadata

Many major content production facilities have custom methods for handling the descriptive metadata associated with the content. As audio-visual content exchange moves from tape-based to file-based operations, the opportunity arises to be able to both embed descriptive metadata in the file and to provide an intimate relationship between the metadata and the audio-visual content. This process allows metadata to be accrued within a file as it passes between operations in a way that has rarely before been achieved. Many users see the ability to embed metadata into an MXF file as a key requirement that extends the use of MXF beyond that primarily for the simple exchange of audio-visual content. As a result of this desire, a list of user requirements for descriptive metadata has been developed as below.

SMPTE EG 41 (MXF Engineering Guideline) gives a table of user requirements for file formats as developed by the EBU. Although not all entries are valid for descriptive metadata, the following may apply:

NOTE – The following priorities were assigned:

A = Baseline ("Must"), B = Enhanced ("Can"), C = Extended ("May") U = Undecided or not determined, X = not allowed (should not be allowed).

Table 1 – User requirements table for descriptive metadata

Priority	User Requirement	**Publication** (Emission, Transmission, Store & Forward, etc.)	**Content Repository**	**Finished Interchange**	**Authoring Interchange**
A List					
A++	Must be easy to understand & apply and standardized	Y	Y	Y	Not easy
	Low implementation overhead	Y	E.g. Could be complex if editing required	Y	No
A	Must be open (as per ITU definition)	Y	Y	Y	Y
A	Must provide Identification of the metadata scheme	Y	Y	Y	Y
A	Must provide for normative templates	Y	Y	Y	Y
A	Must be extensible in header and body (by KLV coding?) (E.g. from one frame to many frames)	Y	Y	Y	Y
A	Scalability (small file/single frame to large file)	Y	Y	Y	Y
A	Must provide synchronization for multiple essence types e.g. Audio/Video/Data Essence and certain Metadata	Y	Y	Y	Y
A	Must be usable on major platforms/OSs	Y	Y	Y	Y
A	Must be application independent	Y	Y	Y	Y
A	Must be transport and storage mechanism independent	Y	Y	Y	Y
A	Simple and complex template (backward-forward compatibility?)	Y simple	Y Both	Y simple	Y complex

Priority	User Requirement	**Publication** (Emission, Transmission, Store & Forward, etc.)	**Content Repository**	**Finished Interchange**	**Authoring Interchange**
A	Extensibility to include non-predefined data (e.g. dark Metadata)	Undesirable	Undesirable	Undesirable	Y
B List					
B	Link Metadata to structural composition information	N	Y	Maybe	Y
B	Can accommodate a range of GoPs (e.g. MPEG)	Y	Y	Y	Y
B	Assignable granularity of Metadata (field, frame/clip/file)	Y	Y	Y	Y
C List					
C	Extensible for internet. Metadata as binary and text format	Y	Y	Y	Y
C	Allow externally referenced essence files for certain applications such as Archiving. A proper standardisation/documentation is prerequisite if external references are used.	N	Undesirable	N	Y
X/U List					
X	Allow proprietary vendor created templates	?	?	?	Maybe

5 Relating descriptive metadata to essence

The MXF format specification includes a standardized mechanism to link descriptive metadata to any individual essence track, a defined group of essence tracks or all the essence tracks. Furthermore, this mechanism defines the start and stop points along the essence track(s) where the descriptive metadata applies.

5.1 Provisions of the MXF structural metadata

The MXF structural header metadata describes the audio-visual content divided into individual essence tracks. Thus each track has associated with it a descriptor of the essence, where each property of that descriptor is static for that track duration. All the descriptors defined so far are file descriptors. In MXF, each track may be divided into one or more segments, where each segment comprises a link to the appropriate segment of essence in the MXF essence container. This applies whether or not the essence is embedded in the file body. The same principles apply for descriptive metadata tracks.

MXF has extended the AAF object model to introduce descriptive metadata tracks that can be used to describe the content of the essence container. Additional objects include a “DM segment” that defines a descriptive metadata track that can be used to describe the essence tracks and a “DM source clip” that can be used to link to either a real metadata track in the essence container or to a descriptive metadata track in the header metadata.

NOTE – Examples of a metadata track within the essence container are defined in the system item of the SDTI-CP specifications (SMPTE 326M and SMPTE 331M).

5.2 DM tracks

Descriptive metadata tracks may be timeline, event or static. The MXF format standard defines the use of these track types for descriptive metadata. This use is summarized as follows:

5.2.1 Timeline track

There may be multiple DM segments on a timeline track and the sum of the segments must equal the total track length. Thus adjacent segments of descriptive metadata will be butt-joined together and leave no gaps. It is perfectly valid to have a timeline track with one DM segment which occupies the whole track duration.

A timeline descriptive metadata track would be used to support metadata that is effectively timed along the track in a piecewise linear manner.

5.2.2 Event track

There may be multiple DM segments on an event track and the segments are unconstrained along the time axis. Thus adjacent segments may overlap or they may leave gaps where no descriptive metadata applies. This is the default kind of descriptive metadata track and allows the greatest flexibility as illustrated in Figure 1.

5.2.3 Static track

If the track is a static type, then it has no timeline and all DM segments apply descriptive metadata to the entire duration of the linked essence track(s). There can be multiple DM segments that link to different essence tracks, but all DM segments in a static track describe the contents of the entire track duration. This is a special case of using descriptive metadata and applies to those kinds of descriptive metadata that are inherently static for the duration of the essence.

Descriptive metadata on a static track cannot be accessed directly if the essence has a timeline. In this case, the material package would access the static descriptive metadata in a source package using a DM source clip on a timeline or event track. This is analogous to sourcing a JPEG still image on a static track in a source package from a timeline track in the material package. Accessing metadata tracks through the DM source clip mechanism is explained in the next section.

If the descriptive metadata is describing a static essence element such as JPEG, then a static track would be used.

5.3 Using DM segments

MXF provides for metadata tracks that logically run in parallel with the essence tracks. In effect, this adds tracks where metadata can be defined that links to one or more frames of one or more essence tracks. Note that MXF uses the same concept for the timecode track with the exception that the time-code track always defines the time for all the other essence tracks.

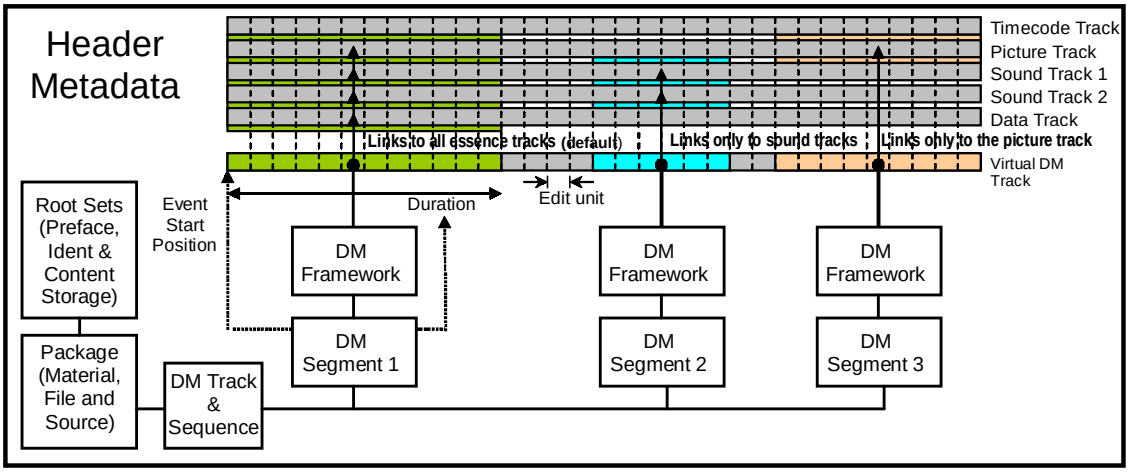

Figure 1 – Descriptive metadata track segments and their relationship to the content of an MXF essence container

Figure 1 illustrates that DM segment 1 is linking the metadata framework(s) of this segment to all the essence tracks of the first part of the A/V content (excluding the time-code track). DM segment 2 is linking the metadata framework(s) of this segment to only the middle part of the sound tracks. DM segment 3 is linking the metadata framework(s) to only the last part of the picture track (excluding the time-code track).

5.3.1 Summary of operation

The description above is of the plug-in mechanism used in the MXF format. It can be summarized as follows:

1. The structural metadata provides a DM track that has a sequence. That sequence has one or more DM segments. Each DM segment defines the start and duration of the DM as well as identifying which essence tracks this DM segment describes.

2. The DM segment also provides a strong reference to a DM framework so that it can '*own*' the DM framework.

3. This strong reference is the point at which the plug-in operates.

NOTE – The start and duration properties in the DM segment mentioned above will depend on the track kind as described above. For example, in a timeline track, the start point of each DM segment is not required because the segments are contiguous.

5.4 Using DM source clips

MXF also provides for sourcing from descriptive metadata tracks that symbolically run in parallel with the essence tracks. A common application of the use of DM source clips in the material package to access DM segments in the source package. In common with essence source clips, DM source clips must use only a timeline track and are not supported for use in event tracks or static tracks.

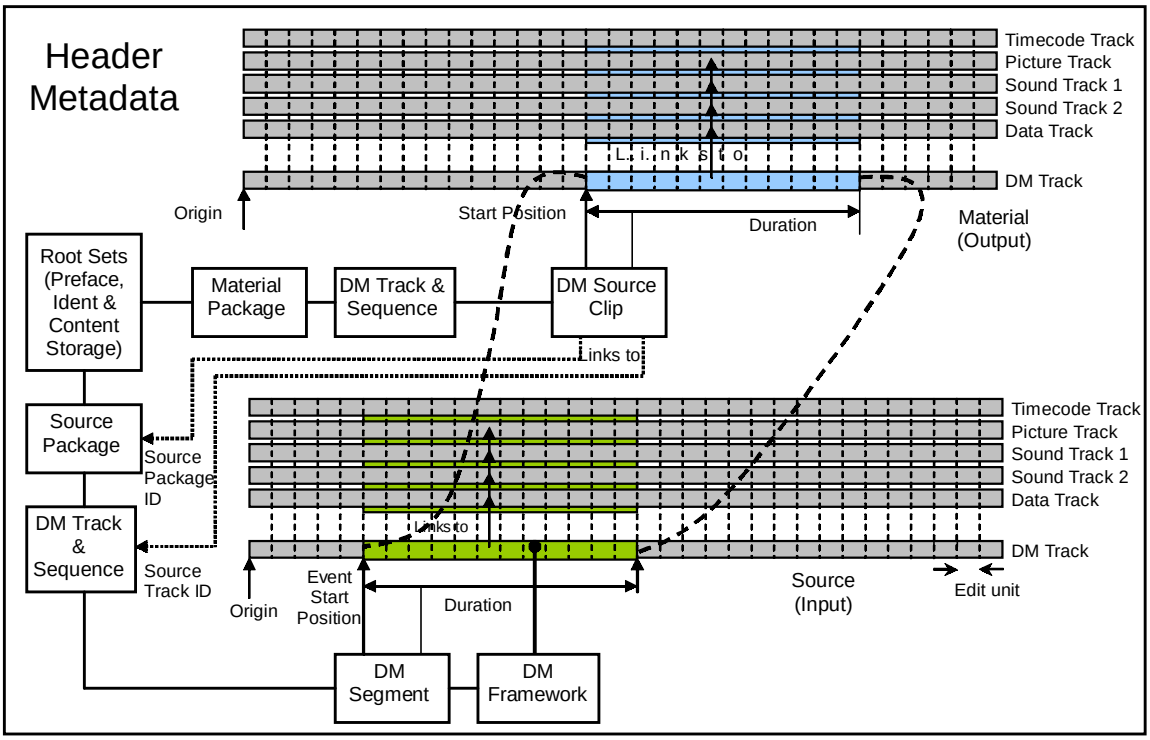

Figure 2 – Descriptive metadata track source clips and their relationship to the content of an MXF essence container

Figure 2 illustrates that the segment of a descriptive metadata track in a source package links the metadata framework to the segment of the essence tracks of the A/V content in the essence container (excluding the time-code track). The source clip of the descriptive metadata track in the material package then references the DM track in the source package and links it to the A/V content in the playout timeline. This referencing removes the need to duplicate the DM framework in the source package.

Furthermore, if the essence container included a true metadata track (such as a system item in SDTI-CP) then the DM source clip could be used to reference the track ID of the actual metadata track.

Note that if the clip is from the current package, the SourcePackageID has the value of the current PackageID.

5.5 Identification through the UL

The MXF format provides the template for a SMPTE label (SMPTE 298M) to identify the DM scheme in use. This is recorded in the preface set of the structural metadata so that decoders may have an early warning that descriptive metadata exists and understand whether it is a recognized scheme.

5.5.1 Identification of MXF descriptive metadata schemes

MXF DM schemes are those standardized schemes which use the plug-in mechanism provided by the MXF structural metadata as described above. MXF DM schemes must be designed within a known set of constraints, as described in section 8. The MXF format specification provides the template for the UL (or ULs) that identifies the scheme in use and the values for bytes 13~16 must be publicly registered in order that the scheme may become known to the decoder. Every decoder will know that the scheme is public because

bytes 1~12 are fixed for all MXF DM schemes. However, a decoder response to the descriptive metadata will depend on whether it understands the particular scheme identified.

Each local set that implements any MXF DM scheme has bytes 1~12 of every set key defined by the MXF format specification. Byte 13 of every such key is defined to have the same value as that in byte 13 of the UL that identifies the scheme hence multiple MXF DM schemes can be accommodated in the MXF header metadata area.

5.5.2 Identification of non-MXF descriptive metadata schemes

Non-MXF descriptive metadata schemes are those schemes which use ULs and keys other that those defined by the MXF format specification. The UL for such schemes must differ in bytes 1~12 in order that they are recognizably non-MXF schemes. Furthermore, the keys used must all differ from those defined for MXF DM schemes.

6 Issues relating to MXF and AAF

This section outlines some of the considerations that need to be made in creating a MXF and AAF compatible DM scheme.

Annex B describes some metadata modeling techniques that may be used to develop descriptive metadata.

6.1 Issues relating to MXF

6.1.1 Using the instance and generation UID properties

MXF DM schemes should use the “instance UID” property as defined in the MXF format specification to provide an object ID for referencing between objects.

MXF DM schemes should also use the generation UID property as defined in the MXF format specification (section 6.6.2) to link to the identification set that was created at the point of creation or change. MXF DM schemes can use the same mechanism for identifying changes by adding the optional generation UID property to each descriptive metadata set.

The UL and local tag values of both the instance and generation UID properties are defined in the MXF format specification.

6.2 Issues relating to AAF

MXF descriptive metadata can be used by AAF implementations if it follows certain rules. Annex B of the MXF format specification defines the DM sets which MXF descriptive metadata can use in order that it can be made visible to both MXF and AAF implementations. Note that not all AAF implementations will support the encoding and decoding of MXF descriptive metadata.

The MXF format specification provides a “DM segment” set (described in section 5.2) which can be used to (strongly) reference a descriptive metadata framework. This framework and all its metadata thus relates to the package, the tracks and the timing defined for the segment. This set is new to AAF version 1 and is a sub-class of the AAF “comment marker” class.

The MXF format specification further provides a “DM SourceClip” set (described in section 5.4) which can be used to link to a metadata track in another package. This is a very useful set in that it allows a copy of descriptive metadata to be made as an alias rather than a simple duplication. Where this method is applied, it can save on unnecessary duplications of descriptive metadata. However, when processing header metadata, the alias link must be followed in order to derive the actual location of the descriptive metadata. The DM SourceClip is a sub-class of the AAF “source clip” class.

6.2.1 Single inheritance hierarchy

The AAF data model does not make use of multiple inheritance. Properties are defined in only one class, and may be used only in instances of that class or its subclasses. In AAF compatible DM schemes, this means that whenever the same property name, UL, or tag appears in more than one set specification, it can be inferred either that one set is derived from the other, or that both are derived from a common abstract superclass.

When it is essential to map metadata from schemes that use multiple inheritance to an AAF compatible DM scheme, certain techniques can be used to maintain the single inheritance requirements. These include the use of new classes that contain one or more properties that can be instantiated multiple times with each instance being the target of a strong reference from the referencing set. An example of this can be found in the use of the name-value set of SMPTE 380M (MXF DMS-1).

Another approach is to define the properties that are needed by several classes in a common superclass. The superclass must be sufficiently high in the single inheritance hierarchy to be available to all subclasses that require the properties. The value of one property (or presence or absence of an optional property) may be used to control the use of the inherited properties in particular subclasses. An example of this technique can be found in the SourcePackage definition in SMPTE 377M which may be interpreted as a file package or a physical package dependent upon whether the descriptor is a file descriptor or a physical descriptor.

6.2.2 Need for unique property values

A single inheritance hierarchy requires that each and every property used within the MXF header metadata must be uniquely defined within the overall class hierarchy. This means that each property must have a single location within the overall class hierarchy of the combined MXF structural metadata and DM scheme(s).

Note that all local tags and their respective UIDs that are in use in any partition must be recorded in the primer pack. Thus if there are two DM schemes in use, any properties using the same UL in more than one scheme must use the same local tag value.

Guidance for the handling of multiple DM schemes in a single MXF file is given in section 8.3.

6.2.3 References are unique, too

The need for a single inheritance hierarchy means that both strong and weak references to any set must be unique, too.

Thus, if both set ‘A’ and set ‘B’ references set ‘C’, the individual reference from set ‘A’ must have a different value to the reference from set ‘B’. This leads to “typed” references. This need for typed references can have an effect on the data model.

The requirement for a single inheritance hierarchy means that set ‘A’ and set ’B’ both need a reference to the new set ‘C’, but that reference must be unique and cannot be the same in both set ‘A’ and ‘B’. This is similar to defining two individually ‘typed description’ properties.

7 Metadata coding

The data structures considered in the previous section need to be coded into some format for real use. MXF codes everything as KLV (key-length-value) packets. However, descriptive metadata structures can also be coded by other methods. Of particular note is XML (eXtensible Markup Language) now in common use. XML provides the ability to easily understand the data where KLV is more efficient but impractical to read. A technique in use today is to code MXF implementations using an XML input file to a coder that reads the XML data and converts it into the KLV coding required for MXF header metadata.

7.1 Coding in KLV

A review of KLV coding was published in the SMPTE Journal [9] and is recommended as an introduction to those unfamiliar with the SMPTE preferred form of coding.

A framework is coded as a sequence of KLV coded data sets connected by strong (and weak) references. Each KLV set is a local set that uses 2-byte tags and 2-byte local tags.

An example of the implementation of a framework is shown next in Figure 3.

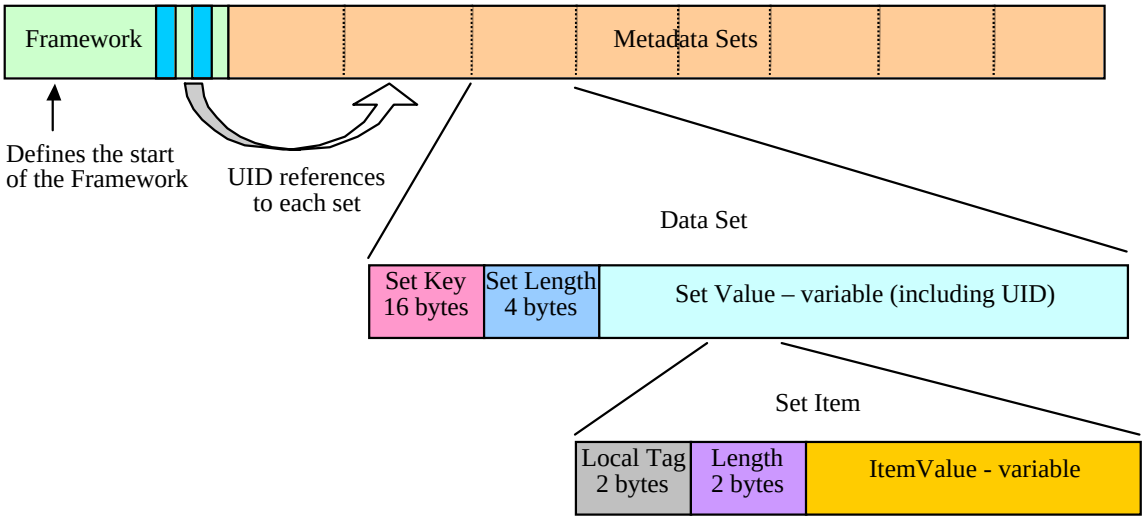

Figure 3 – Three-level KLV data construct of a framework

7.1.1 Using the primer pack for local tag assignment

The MXF format specification provides for a primer pack that identifies all the local tags used for local set coding in MXF. Each local tag must be unique for each property and is an alias for the globally unique 16-byte UID, either as a publicly defined SMPTE metadata dictionary UL or as a privately defined UUID. According to the MXF specification, local tags are unique within the partition in which they are used. The primer pack is provided to ensure that local tags are unique whether they be publicly declared (as per the MXF format specification) or privately assigned.

All local tags for any MXF DM scheme must be dynamically allocated for each partition. This means that, at each encoding, any local tag for any descriptive metadata property must check first with the primer pack to ensure that both the local tag value and the UL have not previously been used.

The following rules apply to all DM schemes that use sets with 2-byte local tags.

1. All local tags for descriptive metadata must lie in the range ’80.00h’ to ‘FF.FFh’.

2. For all statically allocated local tags, the associated UIDs are defined for structural metadata or index tables and cannot be used for descriptive metadata unless the property is inherited through the class inheritance mechanism. Where such properties exist (e.g., instance UID and generation UID), the local tag values are per the statically allocated local tags.

3. For all dynamically allocated local tags, the associated UIDs must be unique within the scope of the primer pack.

7.1.2 Using strong and weak references

The use of strong and weak references in MXF is fully described in SMPTE EG 41. DM schemes are encouraged to use the same set linking mechanisms as defined for MXF structural metadata for consistency.

7.2 Coding in XML

MXF files encode all essence and metadata in its native KLV format. From the beginning the need for a text-based representation of MXF header metadata has been recognized. The main requirement is that it reflects the structure of MXF metadata. The XML (eXtensible Mark-up Language) is ideally suited for this purpose; being an internationally accepted open standard designed for exchange of structured text and data that is intended for dissemination and publication on a variety of media.

XML is actually classed as a meta-language, meaning that it is a language that describes other languages, which is ideal for defining a metadata language scheme. XML is defined by the “Worldwide Web Consortium” (see http://www.w3.org/XML).

XML has been used in certain MXF developments to provide a text-based interface for the header metadata of an MXF file — both structural and descriptive metadata. XML is attractive for use in MXF header metadata because the complex data structures can be relatively easily defined and certain XML tools can be used to validate and cross-check the data. However, XML gives no advantage for the coding of essence data or other binary information, such as partition packs and index tables.

The advantages of XML, as a text-based meta-language, are as follows:

1. It can be used to define the XML form of the registry (e.g. tag names etc.).

2. It can be used to convert the XML registry as data in a table form using the XSLT (XML style sheet transform) programming tool (freely available).

3. It can use the XSLT tool to convert the XML registry to other formats such as HTML for use with a web browser.

4. It can use the XSLT tool to convert to ‘C‘ code to ensure that programmers don't make implementation mistakes.

5. It can provide syntax verification and validation of related XML documents.

To operate with MXF descriptive metadata that is natively KLV coded, XML requires bi-directional conversion between XML files and the KLV encoded data.

Some DM schemes may be directly embedded as XML natively stored in the file. However, such XML-based schemes, like other non-KLV schemes are classed as non-MXF and no conversion to KLV is required.

7.2.1 XML coding of MXF metadata

The XML is highly flexible and allows for many different ways of encoding and structuring information. For instance, data can be stored either as XML elements or as XML attributes to defined XML elements.

As an example: two valid representations of an XML element called “Person” are:

```
<Person>
    <name>Peter</name>
</Person>
```

and

```
<Person name=”Peter”/>
```

Although their meaning is similar, these representations are fundamentally different. As a consequence applications have to process the two XML “styles” in different ways. In order to create platform and application independent XML documents and structures, it is desirable to agree on one common format for XML documents.

7.2.2 XML-DTD v XML-Schema

In contrast to XML-DTD documents, XML-Schema documents follow the XML syntax and can therefore be used by standard XML processing tools. XML-Schema also provides more structural and design flexibility (see the web-link http://www.w3.org/XML/Schema for more details).

Annex C describes an example implementation of an XML-Schema for MXF development.

8 Guide for implementing descriptive metadata schemes in MXF

8.1 MXF descriptive metadata schemes

If a MXF DM scheme implements a single inheritance hierarchy, ensures that its properties are unique within the scope of AAF and provides a suitable class hierarchy, then it may be used by appropriately updated AAF tools as an extension.

Section 6.2 provided the detailed information needed to create MXF DM schemes that may be used by AAF tools.

INFORMATIVE NOTE – Further guidance on implementing descriptive metadata may be found on the MXF Implementers Group web site at http://www.pro-mpeg.org/ or http://www.mxf.tv/.

Below is a summary of the key points in order to implement a MXF DM scheme that can be used by AAF decoders as well as MXF decoders:

1. Ensure that the descriptive metadata is referenced by either the DM segment or the DM source clip set in order that an AAF-aware application can use it as a starting point.

2. Ensure that an appropriate SMPTE UL is defined and that it is present in the preface set of the MXF structural metadata. This common location ensures that the decoder can be sure to parse that UL to check if the scheme is within its repertoire of decodable schemes.

3. Ensure that bytes 1 to 12 inclusive of the SMPTE UL and of every set key is defined as per the SMPTE Format specification (section 8.7).

4. Ensure that the scheme is assigned a registered value in byte 13 of the UL and of every set key. The range of values available is limited to 02h to 7Fh.

5. Ensure that the scheme is defined by a single-inheritance class hierarchy as described in section 6.2.1 of this document.

6. Ensure all property UL values and names are unique within the scope of the AAF metadata and within the scheme itself.

7. Ensure all property UL values are registered in the SMPTE metadata dictionary (this ensures all properties are globally unique).

8. Ensure that the number of properties does not exceed 32K (the capacity of the primer pack for dynamically assigned local tag values). Note that, where multiple DM schemes exist in a file, they will have to share the 32K space assigned for dynamically assigned local tag values.

Note that although different MXF DM schemes can be easily separated by the value in byte 13 of a set key, the sets that should be parsed are those that are the target of a strong reference from within the framework of a DM scheme. This includes sets that may be dark to a defined DM scheme.

8.2 Non-MXF descriptive metadata schemes

Non-MXF DM schemes do not use the DM UL and key values defined in the MXF format specification.

Non-MXF DM schemes can be divided into two broad categories:

1. Those that use the plug-in mechanism described in section 5. These may use any metadata track kind (static, event or timeline) with each DM segment strongly referencing a single KLV block which may be a KLV item, set or pack of any kind defined in SMPTE 336M. The referenced KLV block may reference further KLV blocks as required by the non-MXF DM scheme. It may also contain other forms of coding; for example the value in the KLV packet may be a single HTML data file or a set of XML data files.

2. Those that are present in the MXF header metadata area, but do not use the plug-in mechanism. The outer layer of coding must use SMPTE 336M KLV coding to comply with the MXF format specification, but the contents of the KLV packet may be any data format as identified by the key value. Since this non-MXF DM scheme has no reference to the essence via the DM track/sequence/segment plug-in, it must provide its own internal references where needed.

Within the outer KLV packet, non-MXF DM schemes may use non-KLV coding, or may use KLV coding options other than those specified for MXF header metadata. Non-MXF schemes may also use different methods of set referencing constructs. They may use other forms of coding such as XML.

All non-MXF DM schemes should be identified with a unique ID whose value(s) should be entered in the “DM schemes” property of the preface set.

8.3 Multiple DM schemes in MXF

Files may be created where the single inheritance rule is preserved in the header metadata and there are multiple DM schemes present. Under these circumstances, a metadata decoder that requires single inheritance is likely to be able to unambiguously extract all the metadata within all the DM schemes in the header metadata.

Files may be created where the combination of multiple DM schemes results in conformance to the single inheritance rule being unknown. A cautious approach is to individually extract the metadata from one DM scheme at a time. This will reduce the chances of conflict between properties defined within different DM schemes.

9 Operational considerations

9.1 Using descriptive metadata in file recovery and partial restore

There are no prescriptive solutions for the recovery of file data either as a result of data corruption or to support specialist operations. The two subsections below provide some initial guidance for the designer. This guidance does not limit the scope of what can be done but aims to provide a starting point for creative solutions.

9.1.1 File recovery

SMPTE EG 41 (section 8.2.6.1) describes how body partitions can be embedded in a file at regular intervals to provide the ability to recover essence and metadata in the event of a damaged file. In order to recover metadata, whether structural or descriptive, each partition should include header metadata. In the cases of

body and footer partitions, the header metadata will ideally be repetitions of that present in the header partition. In cases where the header metadata has been built up ‘on the fly’ and never updated on file closure, then the last partition will likely carry the most complete metadata.

In the event of the loss of part of a file, SMPTE EG 41 illustrates how partitions can be reliably recovered from a damaged file. Once partitions have been recovered, the header metadata may then optionally be edited to properly describe the recovered file contents.

It is recommended that each current top-level file package be converted into a lower-level source package and that the material package is used as the basis to create a new top-level file package. A completely new material package will be required. This assignment of new packages will create new package ID values so avoiding duplication of IDs that describe different material.

The new top-level file package should reflect the status of the recovered essence by including copies of the relevant essence descriptors and should recalculate the essence track sets to reflect the starting points and durations of the recovered essence.

9.1.2 Partial restore

SMPTE EG41 (section 8.2.6.2) describes how specific portions of essence can be extracted from a MXF file using the RIP and index tables. This section provides guidance for the extraction of both structural and descriptive metadata.

A file that is subject to partial restore should be closed and contain closed and complete header metadata in either in the header partition or the footer partition. From whichever partition is able to supply closed and complete header metadata, the metadata should be extracted for analysis and processing.

That structural metadata which describes the partially restored essence should return both the essence descriptions and any descriptive metadata. In the case of descriptive metadata, that which describes the essence as a whole should be used in full. In the case of descriptive metadata that describes part of the essence, only that which applies to the whole essence or to that part of the partially restored essence should be used. Descriptive metadata that does not relate to the partially restored essence may be discarded.

Annex A (informative)
Descriptive Metadata Characteristics

A.1 Descriptive metadata criteria

Different criteria need to be applied when defining file-based and server-based metadata. These criteria are partly to do with the characteristics of the metadata, and partly to do with the way files are created and managed.

The initial criteria for defining the metadata to be carried within an MXF file were partially derived from example card indexes and tape labels; namely information which is perpetually durable, finitely durable and public. To assist with understanding where metadata is best located, its characteristics must be understood.

A generalization of metadata characterization has resulted in two broadly independent axes as illustrated below:

	Persistently Durable	Finitely Durable	Transient
Static			
Dynamic			

Figure A.1 – Metadata axes

Each axis will now be described immediately below and in further detail in annex A.2 and A.3.

A.2 Metadata dynamics

Metadata can be broadly characterized as static or dynamic. Static metadata is that which applies to an item of content and dynamic metadata is metadata that has changing values within the content. Within this broad characterization lie a number of further sub-divisions.

A.3 Metadata durability

Metadata can be broadly characterized as durable or transient. The premise is that durable metadata in a file should be unaffected by any exchange or further processing unless specifically up-dated. In contrast, transient metadata (such as used for business transactions) must reflect the variable and topical values used in server-based environments in order to serve a wide range of purposes. Such transient metadata will have a wider range of senders and recipients, both human and electronic.

A.3.1 Durable metadata

Durable metadata may be further characterized into ‘perpetually durable’ and ‘finitely durable’ metadata. Perpetually durable metadata is that which is true for all time, such as the location at which a recording was made. This can never change. Finitely durable metadata is that which is correct at the time of writing, but may become stale after a period of time; for example, the address of a person or organization. This is likely to be stable, but at some time in the future may have changed.

Further details and some examples are given below.

A.3.2 Transient metadata

MXF may be used as an interchange mechanism for the carriage of transient metadata, but this metadata should not be retained.

(NOTE – Transient - “Passing with time”).

A.4 Managing metadata

File header metadata should be either perpetually or finitely durable so that it may then be used to support further file processes, and/or imported into a separate database. Finitely durable metadata values cannot be guaranteed to be valid if other operations indirectly affect the accuracy of the metadata values within the file.

The recipient (organization/person/application) must assume that finitely durable metadata was deemed accurate at the time of sending or persisting. But it has to be considered ‘at risk’ in determining whether any metadata retrieved is still valid. The MXF header metadata provides Identification sets for identifying ‘when’, ‘who’ and ‘what’ modified any MXF structural metadata. This same identification mechanism may be used by a MXF DM scheme as an aid to resolve any conflicts over the reliability of metadata values.

A.5 Dark metadata

The MXF format specification defines the meaning of dark data (essence data and metadata) and how to handle dark data via the KLV coding syntax.

In MXF, dark metadata is that which is dark to a decoder at the time of decoding. The decoder recognizes dark data as that which has an unknown key value, whether as a set key or as a property key. (NOTE – The primer pack ensures that all 2-byte local tags values used in an MXF file are mapped against a full 16-byte UID value).

An unknown key value can be the result of one of several possibilities depending on whether the key value is publicly or privately registered.

Key value registration operates at several levels:

1. Key values openly available in a public standard. All these key values are public.

2. Key values only available to the members of a public group. These key values are public to the group members, otherwise private.

3. Key values only available between specified parties. All these key values are private.

Clearly, the most successful metadata exchange will be through the use of publicly standardized key values. Additions of new entities to public standards will also be dark to a decoder that has not been upgraded. However, the structure of key values in MXF often provides a limited ability for a decoder to determine if a dark entity is an extension to an existing specification or part of private specification.

A.6 Metadata dynamics

Static and dynamic metadata has several sub-divisions that can be addressed as follows:

1. Static metadata across the total A/V content duration that applies to all the content as a whole, including all the essence containers that may reside in the file. As a consequence, such metadata needs only be applied once in the file header.

2. Static metadata across the whole duration of any individual essence container. Such static metadata will be different for each individual essence container.

3. Static metadata across a defined duration of the A/V content in an individual essence container where that defined duration is shorter than the total duration of that essence container. In this case, any metadata associated with the defined duration is static for that duration within that essence container.

4. Static metadata across any A/V content duration may also be applied to individual tracks or to groups of tracks within the A/V content. Such metadata might indicate the audio channel numbering and individual track sound levels.

5. Dynamic metadata has values that change over time. Dynamic metadata values may vary in a progressive manner (such as time-code), may vary in a gradual manner (such as pan-and-scan vectors), may vary in an unpredictable manner (e.g. object trajectory metadata) or may be stochastic; i.e., present only at key points on the timeline (e.g., scene change metadata). Such dynamic metadata values will vary with the playback time. Dynamic metadata is best embedded with the essence so that it can be closely bound to the essence. However, there are

many systems limitations which will filter such dynamic metadata and, furthermore, it needs to be readily extractable for many system operations. An alternative approach is to bind the dynamic metadata to the essence via structural metadata, as a stream of metadata.

A.7 Metadata durability

Examples of perpetual metadata and durable metadata are given in this section.

A.7.1 Perpetually durable metadata

Perpetually durable metadata derives its durability from the point in the content lifecycle at which it was created. When television rushes are first captured by a camera, certain items of information will be irrevocably true over time (assuming it was correct when first written), for example:

1. A unique ID (if used)
2. Date and time, at the point of capture
3. Location of recording device
4. Slate or script number (if used)
5. Capture or creation device identification
6. Operator identification
7. Organization owning the device
8. Organization employing the operator
9. Names of contributors
10. Nature of the action captured (description or synopsis)
11. Catalogue entry or index terms

Perpetually durable metadata will always be relevant to this original material, no matter how many times it is re-used. Some values may be automatically generated by the capture device, or pre-programmed at the point of creating the file.

A.7.2 Finitely durable metadata

Finitely durable metadata is that which is stable for a certain duration, but which may change during the lifetime of the content. Finitely durable metadata is more likely to be associated with a finished programme or item, or an interactive component.

The rule for finitely durable metadata is that the information should continue to be relevant to the editorial content regardless of subsequent repeat usage or re-packaging. The list might include:

1. Editorial unique ID (if used)
2. Version information
3. Content description
4. Content genre
5. Initial intended audience
6. Catalogue entry (may be different from rushes)
7. In-and-out cues
8. “Acceptance” date and time, or completion
9. First transmission date and time
10. Organization giving first transmission
11. Contact information of contributors
12. Perpetual embargo imposed or compliance constraints (certification)
13. and more ….

Some apparently finitely durable values may be deceptive without usage qualifiers, such as:

1. Editorial group or parent series title
2. Programme title
3. Item title

These are only durable for the specific editorial version at hand and the relationships between the metadata and its associated essence may change such as a programme or item being associated with a different series in a subsequent project.

Finitely durable header metadata values are valid only when output by the sender or persisted on a storage medium.

Annex B (informative)
Metadata modelling

This informative annex provides the underlying principles that can be used to define MXF descriptive metadata from an engineering perspective. This includes the concept of data layering and object modeling. A formal definition of object modeling is beyond the scope of this document but can be found in various reference material such as indicated in the bibliography. This section will consider object modeling only in the scope of its use in MXF descriptive metadata and should not be considered as a complete treatise on this complex subject.

B.1 Data layering

In order to aid the modeling process, metadata within MXF should be layered as follows:

1. **Scheme**: a collection of metadata frameworks which, although essentially independent entities, are related through a common class hierarchy and may also share resources.

2. **Framework**: a collection of related metadata objects (either as sets or properties) using an instance of a defined class hierarchy.

3. **Set**: a collection of properties that contribute in equal measure to an object whose overall value is greater than the sum of the individual properties.

4. **Property**: an individual item of metadata.

5. **Enumeration**: certain metadata properties have defined values that are assigned for semantic meaning. Property enumerations may be numeric (and hence language independent), rigid textual (and language independent) or flexible textual (with values dependent on language, culture, application or industry). In some limited circumstances, sets may also be enumerated although this is not commonplace.

Note that these layers are sometimes considered from bottom to top rather than from top to bottom as defined above.

The relationship between the layers is as follows:

1. A **scheme** may have one or more independent **frameworks**.

2. A **framework** may have one or more **sets**. Where there is more than one set, they must specify their relationship within the framework.

3. A **set** may have one or more **properties**. Where there is more than one property, each property is generally considered equal in weight and of no defined order within the set.

4. Each individual **property** may have particular attributes such as:

   (a) It may have specifically agreed values (either numeric or textual)
   (b) It may have minimum and maximum values (e.g. min= 16, max = 235)
   (c) If a string property, it may have lower and upper limits to the string length

B.2 What does layering achieve?

A single property definition has little meaning. Take property name such as ‘City’. We know the ‘City’ value is “Oslo”. But what is the connection?

A data set gives context to individual properties. The ‘City’ property might be part of an address set that includes:

House Number, Street Name, District/Town, City, Region/State and Country

Now we know that ‘City’ name is part of a useful grouping.

A framework gives context to sets and the properties within those sets. This can define how is the address set used. Is it a ‘person address’, a ‘company address’, a ‘shop address’?

If the set is part of a framework, then the framework structure can define the relationship between the address set and any other sets in the framework.

Thus layering of data gives context that can be applied at each layer of the data model. This guide will later explain how data modeling techniques can provide a formal methodology for implementing set constructs so that the relationships between each set and the data that it contains can be clearly and methodically defined.

B.3 How is layering implemented?

The core components which form the backbone of a metadata model are:

1. **Inter-set Relationships**: data sets can be related to one another using UML techniques. Currently, set relationships are defined on a per-application basis (such as MXF).

2. **Sets Dictionary**: a dictionary of objects as sets. A dictionary of sets is contained in the forthcoming SMPTE Groups Registry.

3. **Metadata Dictionary**: a dictionary of all the individual metadata properties.

4. **Thesaurus (Lexicon)**: a listing of all the permitted values that a property may use together with the semantic definition. In a thesaurus, the values are textual and may be sensitive to language, culture, industry and other dimensions.

B.4 Granularity of metadata sets

We need to define metadata sets with useful purpose. Too fine a set granularity and the number of sets and inter-set connections rises with little benefit (a set with only two attributes is probably of little value). Too coarse a set granularity results in sets with similar attribute groupings leading to the design of multiple sets with only minimal difference of purpose.

Assuming we design a scheme where the sets contain the minimal number of attributes for small set granularity, then we can then use the following useful rule for the instantiation of an object structure:

Rule: Where a set has a 1-to-1 relationship with another set, apply a rule that the set is a composite of the component sets. The composite set is constructed as a single set, but embeds all the attributes from each component set into one.

The advantage is clarity of functionality in real implementations. If needed, the component sets can be always be decomposed into the generalized class structure without loss. However, this technique is not suitable where the relationship requires >1 set instances within a single object. This means that the set contents must be chosen with care and by design.

B.5 Adding context to a data entity

There are several methods which can be applied to define the context of a data entity and all have been used in various applications. These are as follows:

1. **Individual definition** for each metadata element. The SMPTE Dictionary has some by accident (not necessarily by design). An example is the dictionary entry “Object Country Code” which is an aggregation of the values of ‘Country Code’ and ‘Object’.

2. **Element pairing** (adj->noun), e.g. Person (name + role). This is useful in limited applications and provides context by the aggregation of two values. Note that sets provide the most common format for the modelling of data.

3. **Contextual ‘Key’ values** defining the context for all properties in the set through the key-length-value coding construct. This is not compatible with object model approach, as it requires a set key to include a set context identifier. It does have an attractive aspect in that the context of the data set is defined in the Key value and is thus first to parse and easy to manage. However it is to be discouraged because it is an implementation specific technique that does not lend itself well to other implementations. It is best avoided for well-modeled schemes.

4. **Embedding a 'Data Definition'** in a data set. This allows a single object to be implemented as a data set that may be instantiated with different means as required. For example, a 'track' set may be instantiated as a 'picture track, 'sound track" or 'time-code track" as required, by using the data definition value to define the set context.

5. **A Framework**. Each set in a framework is given context by the nature of the framework to which they belong. This provides data modelling in that the framework provides the root definition for all the properties and sets that it contains.

B.6 Object modelling

The use of O-O design is common practice in the design and development of software applications throughout the world. The process has been refined and improved with the merging of various O-O design methodologies to produce the Unified Modeling Language (UML). UML is now the de-facto standard for O-O software design and the majority of software design tools conform to the UML definitions.

For those unfamiliar with UML, references [10] and [11] form an excellent starting point. This section now summarizes the complex subject of object oriented (O-O) modeling for the purpose of catalyzing the vast amount of information on this subject into a summary of those parts needed for MXF descriptive metadata. This section does not seek to address those parts of a system that are application or user dependent.

In MXF, the following relationships are used to relate one data entity to another:

1. **Association**: a structural relationship that describes a bi-lateral connection between two entities where each entity retains its independence.

2. **Aggregation**: an inclusive relationship where a first entity forms a part of a second entity, but where the first entity can exist without the second.

3. **Composition**: an exclusive relationship where a first entity belongs to a second entity and where the first entity may be expected to live and die with the second entity.

4. **Dependency**: a semantic relationship in which a first entity can change the meaning of a second (dependent) entity.

5. **Generalization**: an inheritance relationship where a first entity adds value to a second entity. An important aspect of this relationship is that the child entity can over-ride a property value in the parent entity.

In MXF, **association** is frequently implicit and uses terms such as "links to". An example is where the descriptive metadata can be "linked to" the essence tracks using an array of track IDs (see Figure 1 and Figure 2). **Association** is not typically indicated explicitly in the figures used in MXF.

In MXF, the strongest form of relationship is that of "strong referencing". Another form is "weak referencing" which is subject to two further variants (see SMPTE EG41). Even within the UML community, there are different approaches to this closely related group of relationships and this document does not try to resolve that problem. For MXF documentation of descriptive metadata, it is recommended that "generalized weak references" are used for the **aggregation** of sets and "strong references" are used for the **composition** of sets, with this latter term being considered as a stronger form of aggregation. Fowler [11] illustrates the distinction in chapter 5, page 80~81.

In MXF, the often implied, **generalization** is present as a class hierarchy diagram. The MXF structural metadata is defined by the AAF class hierarchy. For MXF descriptive metadata, a class hierarchy is required only for those schemes that are intended to be compatible with AAF. However, it is to be expected that most DM schemes of any reasonable complexity will use a class hierarchy.

In some MXF DM schemes, **dependency** may be implicit. An example is a set that has a thesaurus property that can be used to define specific values for other properties in the set. This relationship is typically not explicitly defined in MXF descriptive metadata.

The symbols typically used in MXF are shown next in figure B.1.

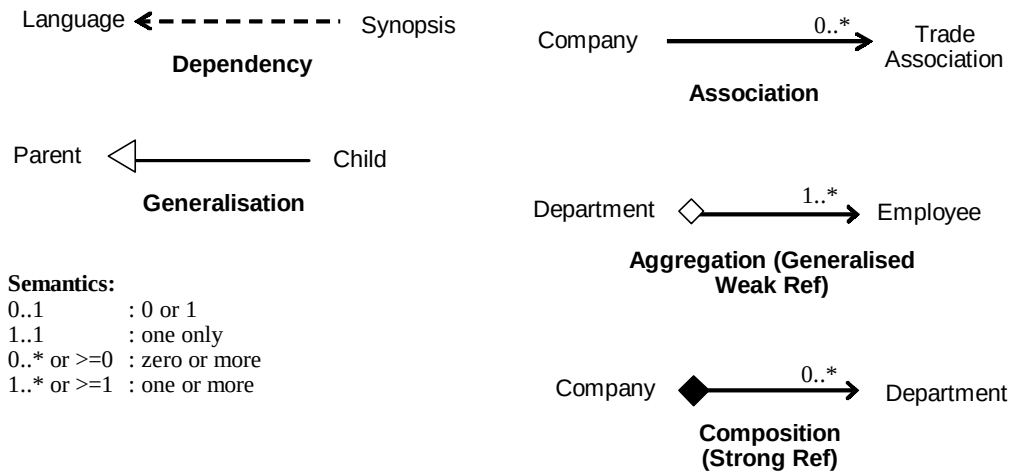

Figure B.1 – UML semantic definitions

Note that there are different conventions for expressing the UML relationships summarized by Fowler [11].

B.7 Class diagrams and object relationships

B.8 What is a class diagram?

In general, a class diagram is an abstract grouping of classes that provides a collective relationship between classes and properties based on the relationships described in the previous section. When any instance of a class is created from its abstract definition it becomes an object with values entered in the property fields.

Individual classes form the basic building blocks of class diagram where, typically, each class represents the minimum data necessary to define the entity desired.

B.9 What is class inheritance?

In many cases, class definitions do not exist in isolation, but form part of a larger model defined by a hierarchy where each class is a sub-class of another class. The starting point is a root class that contains any properties needed by all the other classes in this data model and is the one class that has no superclass. Each sub-class from the root class then inherits any properties defined by the root class and adds its own properties. Within the model, each class only defines each property that it directly owns. However, because of the inheritance hierarchy rules, the class actually contains not just the properties that it directly owns, but also the properties from its superclass and all the other classes higher up the chain to the root class.

In certain cases, classes are defined with no additional properties. Very often, this is a placeholder class, so that it can be an ‘abstract’ superclass for all its sub-classes. The root class is often such an abstract superclass.

When an instance is made from a class, it becomes an object and that object includes all the properties from that class and all its superclasses. This point is illustrated in figure B.2.

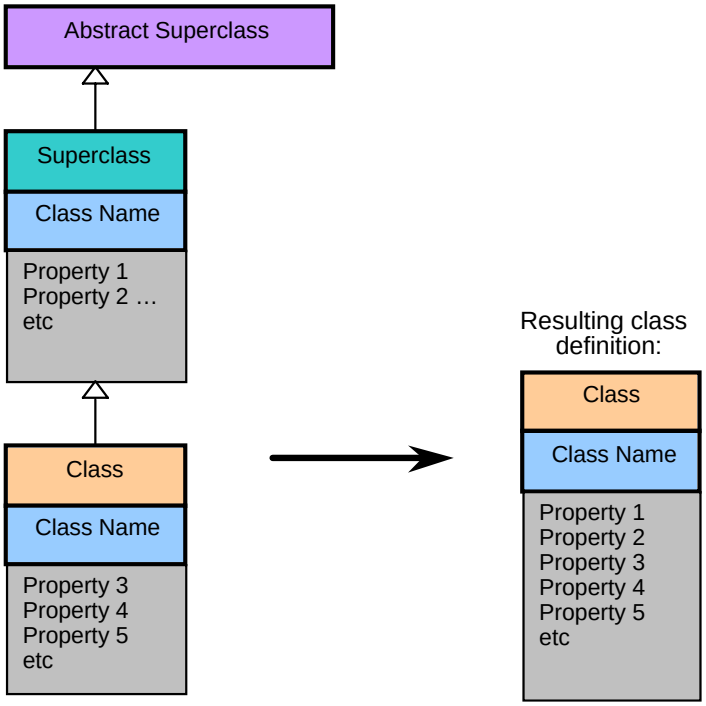

Figure B.2 – Class inheritance and instances

B.10 How are classes used?

As an example, a car seat may be designed as a basic entity to which lumber support may be added as a refinement. A yet further refinement may add electrical adjustments. The most comprehensive might include memory settings for different drivers. In a class hierarchy, this would be constructed as follows:

1. Seat class (superclass of all car seats)
2. Lumber class (adds lumber support to the ‘Seat’ superclass)
3. Electrical class (adds electrically adjustable controls to the ‘Lumber’ class)
4. Memory class (adds memory settings to the ‘Electrical’ class).

Note that a ‘MemorySeat’ cannot be made without defining the ‘ElectricalSeat’ and that an ‘ElectricalSeat’ cannot be made without defining the ‘LumberSeat’.

The semantics of class inheritance is often expressed with the pseudo-word: “IsA”. In the example above, a ‘MemorySeat’ IsA ‘ElectricalSeat’ plus the additional properties defined by the ‘Memory’ class. Following up through the class hierarchy leads to the following statements:

1. ‘MemorySeat’ IsA ‘ElectricalSeat’ plus the individual properties of the ‘Memory’ class.
2. ‘ElectricalSeat’ isA ‘LumberSeat’ plus the individual properties of the ‘Electrical’ class.
3. ‘LumbarSeat’ isA ‘Seat’ plus the individual properties of the ‘Lumbar’ class.

The actual instance of a ‘MemorySeat’ may have blue fabric covers and black electrical seat controls depending on the options available for the value of each property defined in each class.

B.11 Object aggregation and composition

The textbook example is a model of a car where each component of the car may be defined using a hierarchical design. All the cars from the same model are made to the same design, but each car is created as a different ‘instance’ of the

model definition. The model definition includes all the core design parameters which may be static over all instances, or may vary with each instance. The same car may be a blue or red but are still clearly the same car, except the colour 'property' has been changed.

Aggregation and composition are often expressed with the pseudo-word: "HasA". In the example above, a 'Car' HasA 'TurboEngine', a 'SportsBody', four 'AlloyWheels', two 'MemorySeats' and more.

Aggregation and composition can lead to different solutions based on the same overall model. Many major car manufacturers make several models within a range that share many of the same components. For example, the car above might be called a 'sports model'. Another might be a 'saloon model' that "HasA" 'InjectionEngine', a 'SaloonBody', four 'SteelWheels', four 'LumbarSeats' and more.

The creation of each individual car will be an aggregation and composition of all the objects defined by the model, with property values set by the manufacturer or individually as required by the customer.

B.12 Object recursion and its management

There are cases where a generalized data model may lead to a circular reference. Where possible, these should be avoided because the instantiation of the model will result in an infinitely recursive loop.

A classic case is in the definition of a programme. This programme may be composed of several ProgrammeSegments, each of which 'IsA' Programme which may be composed of further ProgrammeSegments and so on.

Figure B.3 shows the aggregation of "wheels within wheels".

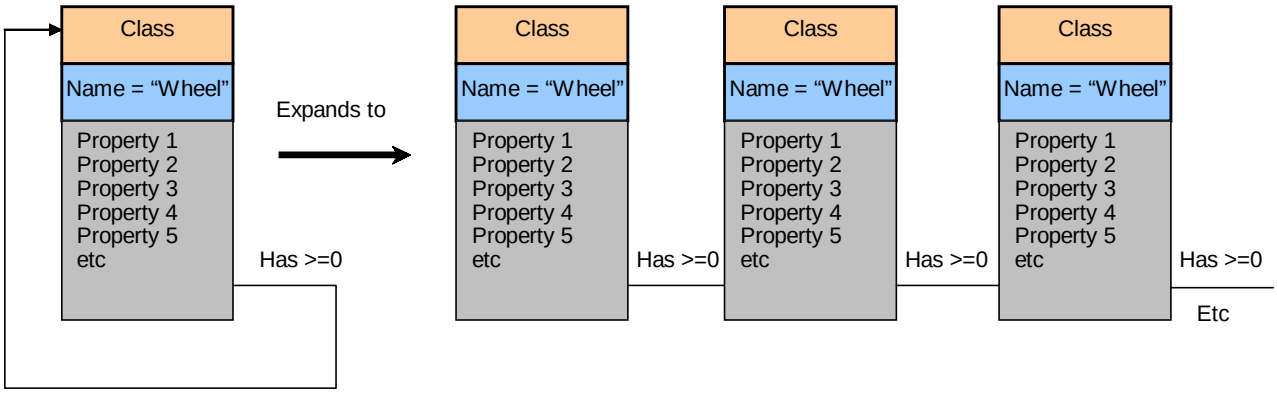

Figure B.3 – Object recursion

Most applications will need to define the limits to any such recursion. Without such limits, decoders may fail, for performance reasons, if they place no limits on the level of recursion supported.

Annex C (informative)
XML-Schema implementation

During prototype MXF developments, there have been several projects using XML for encoding and decoding of MXF header metadata. This annex illustrates one way in which XML was used for the purpose of example. Formal definitions of agreed XML schema may be defined by SMPTE in order to facilitate the interchange of MXF metadata with XML-based metadata systems.

C.1 Example implementation of XML

A set of dictionary files were created, each based on an XML-Schema. An XML-Schema dictionary document was designed for the MXF metadata (including the structural metadata and the DMS-1). All MXF-specific data types (e.g., Universal Label and Universal Label Collection) were defined in a separate XML-Schema document. Because MXF provides for DM scheme “plug-ins”, each metadata scheme (including structural metadata) can be assigned its own XML namespace.

For mapping the MXF metadata structures to XML, the following approach was taken:

- All KLV local sets or packs were encoded as XML elements. All defined KLV items of this local set or pack were encoded as child XML elements. The name of each element and the name of the XML element representing the local set or pack was taken directly from the MXF standard specification. For example:

```
<Person>
    <firstGivenName>Peter</firstGivenName>
</Person>
```

- The key for each local set or pack, as well as the statically defined local tags for each of its KLV items, was stored in the XML-Schema as annotation using the <xsd:annotation> and <xsd:documentation> facilities. Consequently, users did not have to enter the binary key values into their XML metadata documents.

- KLV sets and packs in MXF were modelled on the basis of a class hierarchy. The XML-Schema representations followed this model closely, by defining abstract MXF local sets and super classes as abstract XML elements. MXF data Sub classes were encoded using the substitutionGroup attribute and extension element facility in XML-Schema. An example is given below:

```
<xsd:element name=”Partition” type=”Partition”/>
<xsd:complexType name=”Partition” abstract=”yes”>
. // all KLV items for pack Partition are defined here
.
</xsd:complexType>
<xsd:element name=”ClosedHeaderpartition” type=”ClosedHeaderPartition”
substitutionGroup=”Partition”/>
<xsd:complexType name=”ClosedHeaderPartition”>
        <xsd:complexContent>
                <xsd:extension base=”Partition”/>
        </xsd:complexContent>
</xsd:complexType>
```

- The documentation and annotation facility in the XML-Schema language were used to add documentation to each local set, pack or individual KLV item taken directly from the MXF standard specification. In future this may also be utilised for including international language support.

- KLV strong references were encoded as XML containment. The instance UUID that is used to create a strong reference between two different KLV local sets was not encoded in XML. In the example below the XML element Identification stands for a local KLV set of the same name. It is encoded as child element to a KLV item identifications of the local set Preface.

```
<Preface>
    <identifications>
            <Identification>
```

```
                    <companyName>YourCompany</companyName>
                </Identification>
            </identifications>
        <Preface>
```

Validating XML documents with MXF metadata was not the only purpose for which the XML-Schema dictionaries could be used. The extensible style-sheet language (XSL, see http://www.w3.org/Style/XSL) was utilised to transform XML-Schema documents (or any document with XML defined syntax) to other formats including, for example, source code for high-level programming languages such as C++ or Java. This was successfully implemented for the development of an MXF CODEC and an MXF extension to the AAF toolkit. The benefit of this approach was to greatly assist software maintenance and upgrading of MXF CODECS to higher MXF versions. In addition to source code generation, a set of XSL transform documents were developed to generate on-line HTML documentation or to track changes between the different versions of MXF. This proved to be particularly useful in the MXF specification development phase.

Annex D (informative)
Bibliography

1. SMPTE 380M-2004, Material Exchange Format (MXF) — Descriptive Metadata, Scheme 1 (Standard, Dynamic)
2. SMPTE RP 210, Metadata Dictionary Registry of Metadata Element Descriptions
3. SMPTE RP 224, Registry of SMPTE Universal Labels
4. SMPTE 326M-2000, Television — SDTI Content Package Format (SDTI-CP)
5. SMPTE 331M-2004, Television — Element and Metadata Definitions for the SDTI-CP
6. SMPTE 359M-2001, Television and Motion Pictures — Dynamic Documents
7. EBU / SMPTE Task Force for Harmonized Standards for the Exchange of Program Material as Bit-streams – 1998, http://www.smpte.org and http://www.ebu.ch
8. Advanced Authoring Format, http://www.AAFassociation.org
9. The SMPTE Data Coding Protocol and Dictionaries, Jim Wilkinson, SMPTE Journal, July 2000 Vol. 109, No 7, Engineering Report
10. “The Unified Modelling Language User Guide”, Booch, Rumbaugh & Jacobson, Addison Wesley, ISBN: 0-201-57168-4, 1998.
11. “UML Distilled - Applying the standard object modelling language”, Martin Fowler, Addison Wesley, ISBN: 0-201-32563-2, 1999
12. MXF Implementers Group web site: http://www.pro-mpeg.org/ or http://www.mxf.tv/