Encoding Data into MXF files:

BER and KLV encoding

Paper courtesy Omneon. Inc.

There can be no doubt that the Material eXchange Format, or "MXF", is becoming widely adopted as the format for file interchange and interoperability in TV workflows. MXF offers users significant advantages in file-based operations because vital Metadata is included with the actual essence in the file itself, reducing or sometimes eliminating the need to re-enter Metadata at the various stages in the workflow. There is a misconception that the information in an MXF file (Metadata or Essence) is encoded in such a way as to render the information difficult to extract and unreadable by humans. This is not the case. Whilst at the lowest level MXF does use some encapsulation techniques ("wrapping") when encoding data into a file, these techniques in no way compromise the data elements themselves, which means that its possible to view and extract information using even the most basic of binary viewers (although that would require a lot of manual parsing of the file, which would be fairly tedious).

It's important for engineers to fully understand the techniques involved in the creation and modification of MXF files in order to be able to monitor and analyze the health of any individual clip, but the available texts can be confusing.

This paper will detail the various encoding technologies utilized in the creation of MXF files – specifically, the ISO/IEC Basic Encoding Rules ("BER"), and the MXF KLV wrapping scheme.

Why Does Data Need To Be Wrapped?

First, we should clarify the use of the term "Wrapping". Software uses wrapping techniques to create containers which can hold items of data. A wrapper encapsulates a single data source to make it usable in a more convenient fashion than the original unwrapped source. The term is used extensively in the media industry when talking about media (QuickTime is a Wrapper format, as is MXF). Its important to note that by their very nature, wrappers not directly define the format of the data contained within them – that's usually constrained by some other specifications. When talking about MXF, however, things can get a little confusing: Many experts use the term "MXF wrapped" to indicate that some form of essence has been encapsulated into an MXF file. They will also use the term "KLV wrapped" to talk about lower level encapsulation of an individual piece of data. In this case, "KLV wrapping" is exactly synonymous with "KLV encoding", and it is not unusual for both terms to be used in a single specification

MXF files can be big. Very big – it is not unusual for a single video clip (with audio) to occupy Gigabytes of storage. As in all Broadcast formats, there is a wish to make the decoding of these enormous files as simple as possible, putting more of the processing load on the encoder. There are obvious economic reasons for that: since there are likely to be many more decoders than encoders, it is desirable from a system level to make the decoders as inexpensive as possible for any given task. It should be noted that there are still requirements for different tiers of decoder as material passes through the workflow to the consumer, ranging from decoders that simply synchronize and play the essence (video, audio and timecode), to those which allow applications to modify/edit the essence or interact with the Metadata embedded in the file.

As a format, MXF is intended to fulfill the file access and transfer requirements for many workflows, from the relatively simple, to the quite complex. When the standard(s) for MXF were first being written, it was recognized that this wish could be in conflict with the first paragraph (above): a simple decoder is unlikely to be able to deal with a very complex MXF file, so how do you ensure that it doesn't try? The solution

was to develop a scheme in which decoders could find our very early in reading the process whether or not this was a file that they could deal with. If not, they could then produce an error code, and move on to the next task. The same holds true for Metadata. It can be argued that a simple playout decoder can probably ignore all but the most important structural elements contained in clip Metadata. How does it know whether or not any particular Metadata item is important in a specific application (and ignore the ones that aren't)? These considerations led to the creation of the KLV wrapping scheme used in the MXF file format. KLV wrapping itself is dependent on some other work: it makes heavy use of the ISO "BER" (Basic Encoding Rules) in formatting numerical data, so to fully understand KLV wrapping, we must first examine BER.

The Basic Encoding Rules define a method for encoding which consists of giving a piece of data a label (called an Object Identifier, or "OID"), which tells the receiving computer what kind of data is being transmitted (integer number, real number, Boolean value, string, etc). This label is then followed by a length number, which is equal to the number of bytes immediately following the length number. It is these subsequent bytes, called the "Content Bytes" that usually make up the value of the item being transmitted. BER allows for "short" and "long" versions of the length.

The simple form has the length specified by a single byte, whereas the long form can have the length defined by multiple bytes. For the short form, the MSB of the 1st (and only) byte is set to "0", whereas for the long form, the MSB is set to "1", and the remaining LSBs are used to indicate how many additional bytes are being used for the length value. Thus a short form-encoded value is limited to a maximum payload of 127 bytes.

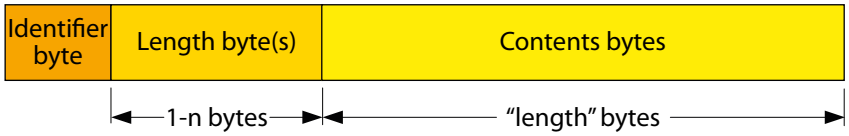

Figure 1. BER coding

ISO/IEC 8825-1: Basic Encoding Rules (BER)

ISO/IEC 8825-1 is a 30-page International Standards document, which details the methods which can be used to pass data from one computer system to another. It is quite broad in scope, and actually covers the Basic Encoding Rules ("BER"), Canonical Encoding Rules ("CER") and Distinguished Encoding Rules ("DER"). MXF files only use the BER, so for this paper we will ignore the other 2 variants (they are merely more restrictive encoding rules, offering less freedom in the encoding process). Fig. 1 illustrates the structure of a BER-encoded value.

For completeness of description, it should be noted that the BER allow for the addition of some "End-of-contents" bytes after the Content Bytes, but this option is not used in MXF files (they're only used if the length of a data value is not known at the start of encoding).

This encoding scheme therefore creates unique strings of bytes, which may represent any kind of item – a name, a value, or even a string of individual bits to be used in some logical activity. The specifications for MXF use this coding methodology extensively, and expand it out to be used in KLV coding.

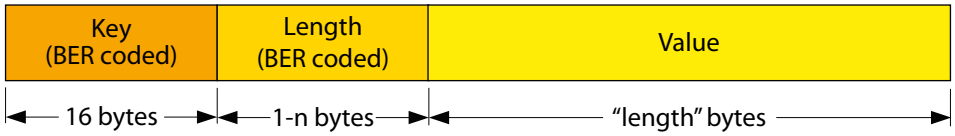

Figure 2. KLV coding

KLV Wrapping

All data in an MXF file, including the essence, is KLV wrapped. At first glance, KLV wrapping looks very much like BER encoding. At the 50,000 foot level, they use very similar structures to encode data. However KLV wrapping offers a great deal more flexibility in the types of data that it can encode: where BER is Typically limited to the encoding of a single data item, KLV can be used to encode an entire set of data values, which offers the ability to encapsulate all of the parameters of an object into a single entity (such as the Metadata for a frame of video, which has height, width, bit depth, etc as its parameter set).

Fig 2 shows the basic structure of a KLV-wrapped piece of data. If you compare it to Fig.1, you'll see that the major difference (at least superficially) is that the single byte Object Identifier (OID) has now been replaced with a 16-byte key.

Key

As you look deeper into the KLV coding, though, you will begin to see some other differences. First, lets discuss the key. The key is actually a Universal Label (or "UL"), as defined in SMPTE 298M. This key is a SMPTE specified 16-byte string, which specifies exactly what kind of data is being carried in the KLV triplet.

The key is itself encoded using BER. The first byte of the key is the OID, and is always 0x06 (for those of you confused by this nomenclature, it simply means the value 06 in Hexadecimal notation).

The second byte of the key (which, referring to the BER description above, is the first byte of the length value for the key) is short form encoded, and always has the value 0x0E. That means that the payload is 14 bytes, so when added to the key OID and length byte, we see that a SMPTE UL is always 16 bytes long.

The second 2 bytes in the key identify this key as being administered by SMPTE (in other words, if you want to look up what the rest of the key means, you need to look it up in a SMPTE document). These two bytes are always 0x2B and 0x34.

Combining the above points means that a SMPTE UL (key) will always start with 0x06, 0x0E, 0x2B, 0x34. Software decoders can look for this pattern to determine the start of any KLV triplet. The remaining 12 bytes make up the detail of the label itself. Finding the meaning of the label requires a dictionary/registry, against which to compare the rest of the label. Byte 5, 6, 7 & 8 of the UL is used to tell you which type of reference document you need to look for (and which version of it applies): a dictionary of Metadata terms (SMPTE RP 210), or a dictionary of labels (SMPTE RP 224). In some circumstances, it can be possible for the definition of the label to tell you how long the payload is. This should not be used as an absolute: that's where the Length parameter comes in.

Length

The Length value performs the same function as the length byte(s) in the BER case. This value is itself BER coded, but the specification mandates that the Length can never be more than 9 bytes in size (which means 8 bytes of payload length). In most cases, real-world MXF files use 4 or 8 bytes to define the size.

Value

Finally, we have the Value. This is where the actual data is placed, and it is inserted exactly as it appears in its original form, in byte order. There is no additional processing applied to the data bytes themselves.

Groups, Sets and Packs

Sometimes, it is convenient to gather together a number of pieces of data into a single logical entity (this is certainly true for metadata items). KLV allows for this by offering the capability to have the data be collected into groups, sets and packs. Outside of convenience, there is another compelling reason for doing this: Its clear that KLV coding offers a lot of flexibility, but it also adds overhead: If the value of the data being carried only fills 2 bytes, for example, the KLV

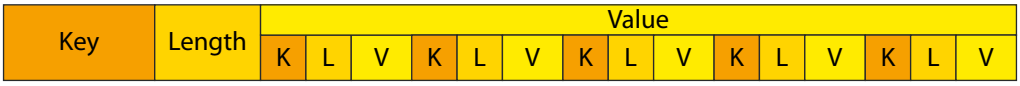

Figure 3. KLV recursion (the "value" is made up of KLV packets

packet will be 19 bytes long – 16 for the key, 1 for the Length, and 2 for the payload. Whilst this might be OK for an occasional small value, if the number of small items is significant (as it is likely to be in the case of Metadata items, which are generally just a few bytes in length) then so is the overhead.

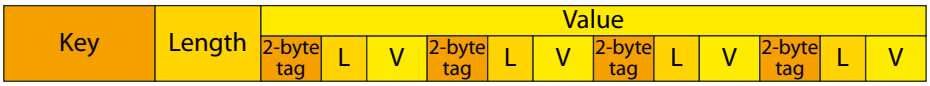

Figure 4. Example of "local set"

Sets

The answers to these problems come in the form of constructs called Universal sets, Local sets, and Variable- and Fixed-length packs – all of which can all be used to improve the coding efficiency. All work on the principle of recursion: You can use one large KLV packet to contain a collection of other KLV packets (see Fig 3). This outer KLV's key indicates that its payload consists of other KLV packets is by a change in byte 5 – which indicates what kind of grouping is being used. It should be noted that while all items in an MXF file are KLV wrapped - including the essence – this recursion/grouping technique is only used for Metadata items (as they tend to have the smaller data sets).

Universal Sets are the simplest construct to understand. They are simply a collection of KLV values sitting under a large KLV umbrella. The only reason for using Universal Sets is to conveniently gather a set of metadata parameters together under a single entity. There is no improvement in efficiency (in fact, its slightly less efficient, as you now have to generate the umbrella Key and Length data). It does emphasize the object-oriented nature of MXF metadata, though, as all of the metadata items for a particular class of object are gathered together into a single KLV item.

Local Sets take the above idea one step further, and add some efficiency in coding. The principle of recursion remains the same, but instead of full 16-byte keys, the recursed data use 1 or 2 byte "Tags" as their Key value. This instantly reduces the coding overhead quite dramatically. Whilst BER coding of the length fields is possible, in practice, the Length fields are generally constrained in size – the recursed KLVs in an MXF local set typically have 1 or 2 byte lengths. The Tags can be converted back into full SMPTE ULs through the use of look-up tables, which are contained in a separate Metadata block called the "Primer Pack", located early in the MXF file. See Fig. 4 for details of a local set.

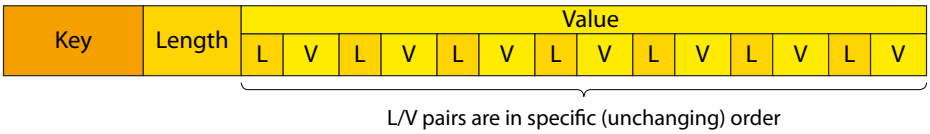

Figure 5. Example of "variable-length pack"

Packs

Variable-Length Packs take this idea further still: a Variable-Length Pack is once again defined in byte 5 of the outer KLV coding. The recursed data elements now have no local tag at all: in a Variable-Length Pack, the data items must exist in a logical, pre-defined order (again, this order is defined in the Primer Pack). See Fig. 5

Finally, we have the option to use a Fixed-Length Pack. In this case, even the length fields are omitted – the specification for the Pack includes the order of the data items, and the length value for each data item. Fixed-Length Packs are analogous to a simple ordered list, contained in a single KLV construct. See Fig 6.

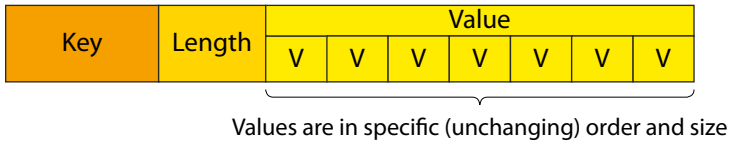

Figure 6. Example of "fixed-length pack"

Conclusion

The combination of Basic Encoding Rules and Key-Length-Value wrapping allows MXF to make use of IT best practices to minimize the effort required to place and organize complex Metadata structures into sensible objects for inclusion with the essence. Whilst initially somewhat daunting, the basic principles are simple to grasp, and once understood, the KLV wrapping scheme can be readily examined for analysis and compliance purposes.

The slight increase in overhead caused by such wrapping is massively outweighed by the improvements in workflow efficiency and management that such techniques enable – applications can be made much more efficient if they have the ability to look at individual data sets and decide if the data carried is important in any particular instance, rather than having to read (potentially) huge amounts of data just to get to the items of interest. This reduces processing time, with a resultant increase in responsiveness, or may mean that less expensive processors can be used to perform a particular task. As files are likely to be accessed many times as they make their way through the workflow, the savings accumulate in each stage, resulting in significantly improved solutions

This white paper was supplied to the AMWA by Omneon, now Harmonic Inc.

Further white papers on MXF, AAF, XML and SOA applied to advanced media workflow can be downloaded from the AMWA website at *www.amwa.tv.*

9/2013