Motion Imagery Standards Board **Recommended Practice** **Common Metadata System: Structure**	**MISB RP 0701.0** **August 6th 2007**

1 Scope

This Recommended Practice (RP) defines the structure of the Common Metadata System (CMS). The CMS is a list of metadata items embedded in KLV data structures that can be used across any sensor/platform and motion imagery system.

There are two sections of the CMS, the structural definition and the content definition. This RP describes the structural definition of CMS and RP 0702 describes the content definition.

This RP describes how to organize the sensor/platform data into a hierarchy of KLV Packs and Local Sets that reduces the bandwidth needed to transmit the data. This RP also defines the required data elements but all other data elements (Content elements) are defined in companion document RP 0702.

This document does not include security structures or content; refer to RP 0102 for security standards.

2 Introduction

The government has designed, built and deployed many different types of motion imagery sensors varying in size, mounting (platform) and purpose. In order to use and analyze the sensor's data some additional information is usually needed that describes, in broad terms, the situational environment that the sensor data were collected in; this situational environment data is called metadata. The situational environment ranges from the position and attitude of the sensor and platform to identifying why the sensor/platform was used or where it was used. The problem to date is that there is little consistency in the metadata provided by sensors/platforms This can and will make exploitation difficult. Each sensor/platform system currently provides its own metadata formats and transmission schemes and each exploitation system has to be customized to process this data. With proper standardization the costs of the sensor/platforms could be reduced and the exploitation systems will not need to be customized for each different platform.

2.1 Background

To date, the primary metadata focus for the MISB has been Unmanned Aerial Systems (UASs); specifically associating metadata with their video. The primary standard used is EG104 (Predator UAV Basic Universal Data Set) that describes how to convert Predator Closed Captioning (CC) data to KLV and it is assumed that the KLV will be muxed into a MISP MPEG-2 stream using the private data stream (PDS) - (MPEG-2 TS w/ KLV PDS). Even though the intent was to use EG104 only for Predator, the standard has been used to encapsulate other types of UAV and sensor metadata into MPEG-2 streams. The fact that the use of the EG104 standard has been stretched for other platforms and sensor types shows that a common metadata methodology is needed.

EG104 is a 1st generation standard that provides a means to communicate the metadata with KLV, however it uses a simple and verbose (large bandwidth) method to format the data.

The intent of this standard is to be a replacement for all existing MISB metadata structure/content standards including EG0601 local data set (LDS), EG104 Predator UAV Basic Universal Set and other implemented proprietary data sets.

2.2 CMS Structure Features

A CMS metadata stream is built from many individual CMS packets. The following outlines the features of both the CMS streams and packets:

1) The CMS includes elements used for verifying the integrity of the data payload (e.g. CRC).
2) The CMS allows metadata to be delivered at a temporal rate determined by the design of the systems producing the KLV – the temporal rates of the data can be independent of the host transport and video stream (*i.e.* independent of the GOP times of MPEG-2).
3) The CMS provides structures to minimize the packet size and the bandwidth used to transmit the metadata.
4) The CMS provides a standardized method for time tagging all CMS Packets as a whole; every packet shall be required to have a timestamp.
5) The CMS provides a standardized method for temporal offsets of different data items. (Data can be measured at different rates – the structure is able to time tag elements with different times.)
6) The CMS provides a standardized method for identifying the metadata stream.
7) The CMS provides a standardized method for identifying the version number of this RP in the stream.
8) The CMS provides a standardized method for identifying the producer (*i.e.* software/company) and version number of the system that has generated the metadata in the stream.
9) The CMS provides a standardized method for specifying the uncertainty of measurements or computations.

3 References

1.1. SMPTE 336M-2007, Data Encoding Protocol Using Key-Length-Value

1.2. RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

1.3. CCITT-16-CRC – Web references:

1.3.1 A Painless Guide To CRC Error Detection Algorithms by Ross N. Williams

1.3.2 CRC16-CCITT - http://www.joegeluso.com/software/articles/ccitt.htm

1.4. MISB: Security RP0102

1.5. MISB: EG104

1.6. MISB: EG601

1.7. MISB: RP0702 – Content part of CMS

1.8. MISB: RP 0603 – Common Time Reference

1.9. MISB: MISP 4.2

1.10. DoD Annex to the SMPTE Metadata Dictionary

4 Common Metadata System: Structure

4.1 Overview

CMS is a KLV construct designed to reduce the bandwidth needed when transmitting data. In the KLV standard (SMPTE 336M), there are provisions for several different ways of formatting metadata; in order from most flexibility (largest size) to least flexibility (smallest size): Universal Sets, Global Sets, Local Sets, Variable Length Packs and Fixed Length Packs. To reduce the bandwidth of the data being transferred, a pack structure is used. In order to effectively use the packs, two new pack based structures need to be defined: Truncation Packs and Floating Length Packs. The Floating Length packs are defined to contain fixed sized elements at the front of the pack followed by a single variable length element. The truncated packs are defined to contain a fixed number of elements however when the pack is formed at “run time” it can be truncated. Details of these new pack constructs are in Section 4.2 below.

With these two new pack structures defined, the CMS packet structure can be described as a combination of KLV Fixed Length Packs (defined in 336M), Local Sets (defined in 336M), Truncation Packs (described below) and Floating Length Packs (described below).

When reporting metadata in the KLV format, a KLV stream is formed by building and sending individual KLV packets of data. Likewise, a CMS packet is KLV data that is built with the CMS structure and a CMS stream is a series of individual CMS packets.

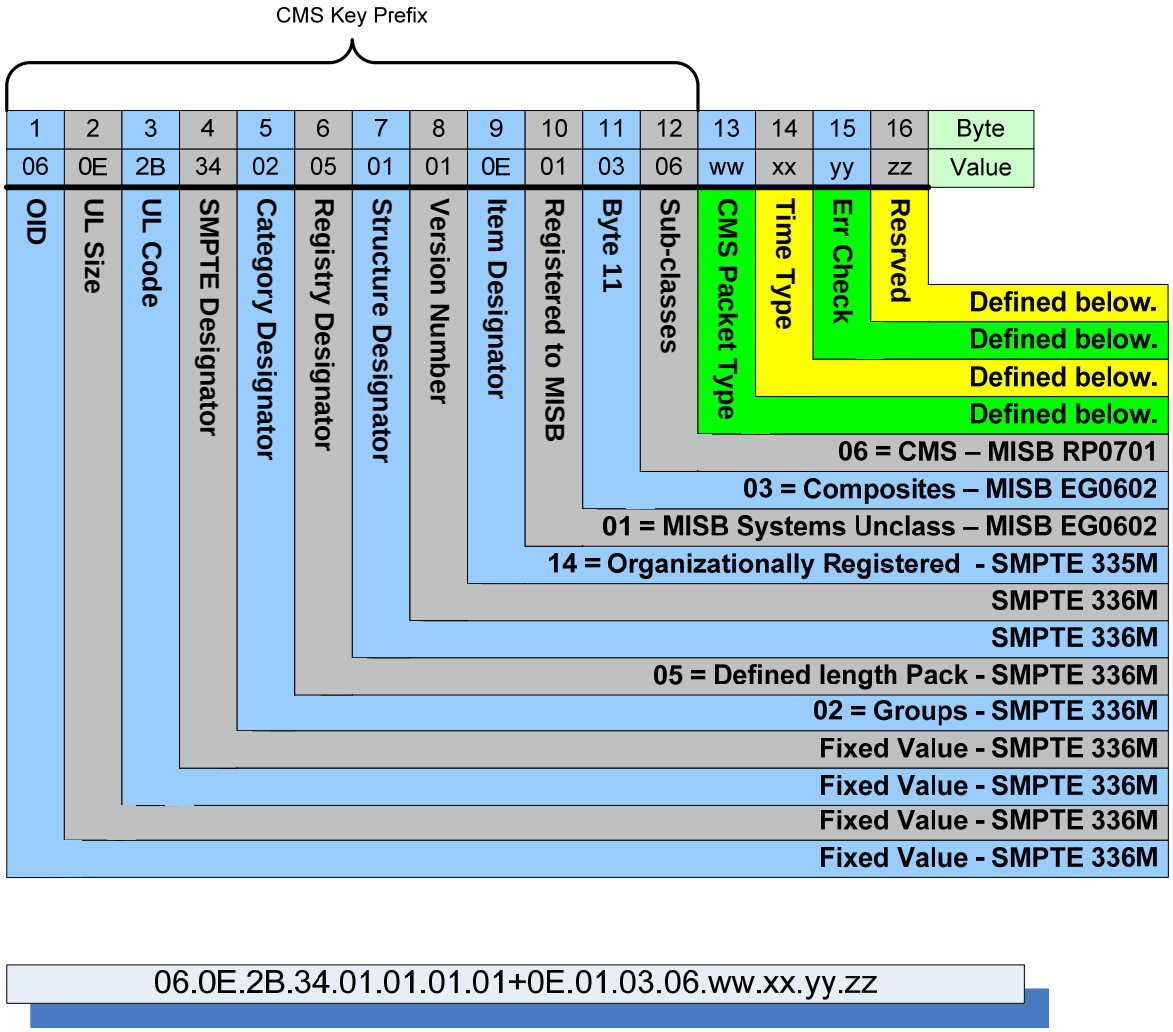
The CMS stream is expected to be embedded along with a video stream (e.g. MPEG2 TS), however, the CMS stream should be able to exist independently from the video or sensor source for downstream analysis or muxing with a video stream. For example, there are systems that use split paths, one for video transmittal and one for metadata transmittal. At the ground station the data is muxed together and re-transmitted. This requires that the CMS packets provide a complete timing infrastructure. For proper muxing of the metadata and sensor data the sensor data needs to have embedded timing information; the standard for embedding this timing information is identified in the RP 0603.

The CMS timing architecture, described in the sections below, provides an absolute packet time along with offset times for individual measured datum values (within the packet). This document describes the base instantiation (or default method) of this architecture by using the microseconds since 1970 time system. Other timing systems may be needed by applications that have requirements for more or less fidelity. To support these other anticipated timing needs, CMS supports variations of the CMS Key. Example of a variant is to have the timing system based on nano-seconds or based on a different start time. These variants must be documented with other standards and registered with the MISB. (Note that changes to timing system may have ripple effects on size of time representation fields, *etc.*)

In addition to the time architecture, alterations of the error checking methods can also be defined. These are also indicated by a variation in the CMS Key.

4.1.1 CMS Keys

CMS relies on a set of CMS keys that indicate different variations of the CMS standards. The base key is defined the DoD Annex to the SMPTE Metadata Dictionary (sometimes referred to as the DoD Dictionary) as:



The first part of the key (called the CMS Key prefix) is composed of 12 bytes and is constant for all packets using the CMS standard.

The remaining bytes are reserved within the DoD dictionary for CMS purposes. The first byte after the CMS Key prefix (“ww”) is used to define the type of CMS packet, either timed (value = 01) or historical (value = 02) – described in Section 4.3.1. The second byte after the CMS Key prefix (“xx”) is used to define the timing system; currently only the base timing system is being used (the system described in this document) the only current value is 01. The third byte after the CMS Key prefix (“yy”) is used to define the error checking method; currently only the base error checking method is used (defined in this document) the only current value

is 01. The remaining byte ("zz") is reserved for future purposes and should always be 00. The following table summarizes this information:

Byte	Description	Valid Values
13 ("ww")	Type of CMS Packet – timed or historical	01 or 02
14 ("xx")	Type of timing System	01
15 ("yy")	Type of Error Checking	01
16 ("zz")	Reserved for future use	00

In summary the following keys are currently defined for CMS:

Key	Usage
06.0E.2B.34.02.05.01.01+ 0E.01.03.06.01.01.01.00	CMS Standard Packet with embedded Timing information – defined in this document.
06.0E.2B.34.02.05.01.01+ 0E.01.03.06.02.01.01.00	CMS Standard Packet for historical data – defined in this document.

4.2 New Pack Constructs

This standard makes use of two new pack types, as explicitly allowed by SMPTE 336M-2007: floating length packs and truncation packs. The following sections provide a quick summary of the fixed length pack as defined by 336M follow by further details on the floating length and truncation pack.

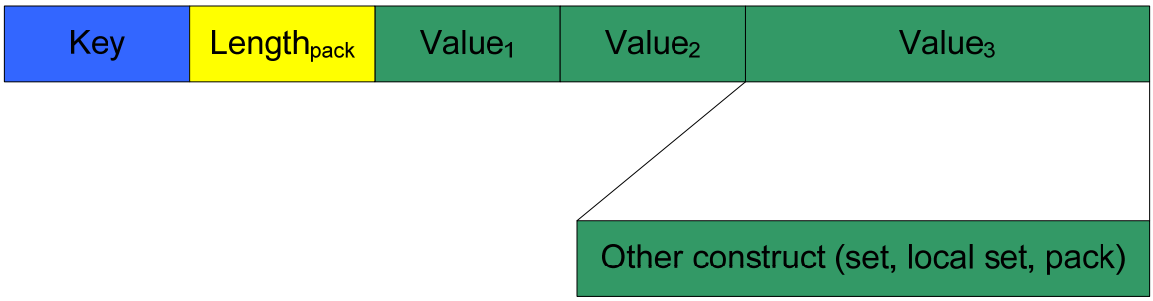
4.2.1 Background: Summary of Definition of standard 336M Fixed Length Pack

Please reference 336M for complete definition of Fixed Length Pack. A fixed length pack definition requires fixed length elements in a specific order; for example a pack would contain elements $Value_1$, $Value_2$ and $Value_3$ – the elements have to be in a pre-defined order and each elements' length (or number of bytes) is predefined – this is equivalent to a fixed length record of bytes. Every pack must have a defining document which describes what each value means; RP 0701 and RP 0702 fully define these packs.

The advantage of using packs is they are the most bit efficient with the smallest overhead. The disadvantage is the lack of flexibility; every value has to be placed in the pack in the exact order as the pack definition. The packs length is fixed based on the pack definition.

4.2.2 Floating Length Pack

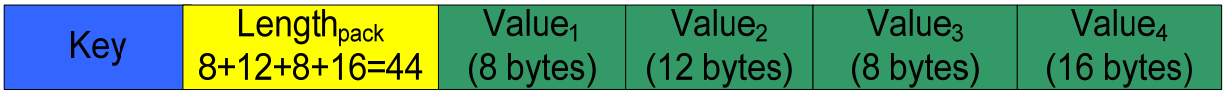
A "floating length" pack is identical to a standard pack except the last elements' length is allowed to "float". This enables the pack to contain another construct within it. The length of the last element is computed by subtracting length of the first n-1 elements from the total pack length. The following diagram shows an example of a "floating length" pack.



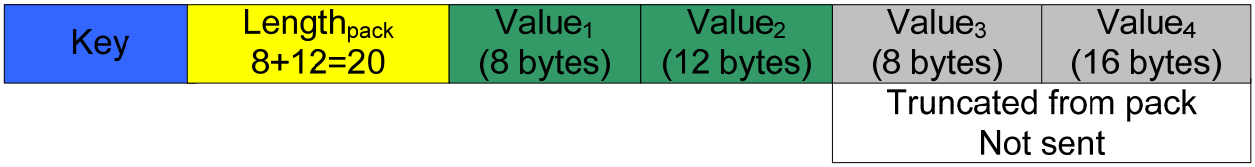
The Key (in blue) is a pack key (standard 16 bytes), the Length$_{pack}$ (in yellow) is the number of bytes used by Value$_1$, Value$_2$ and Value$_3$; Value$_1$ and Value$_2$ are defined with specific lengths in the defining document that describes the pack – Value3 is variable in length and can contain other KLV groups (sets, packs, etc). The length of Value3 is computed by taking the Length$_{pack}$ (in yellow) and subtracting the lengths of value1 and value2: Length3 = Length$_{pack}$ – (Length1 + Length2).

4.2.3 Truncated Pack

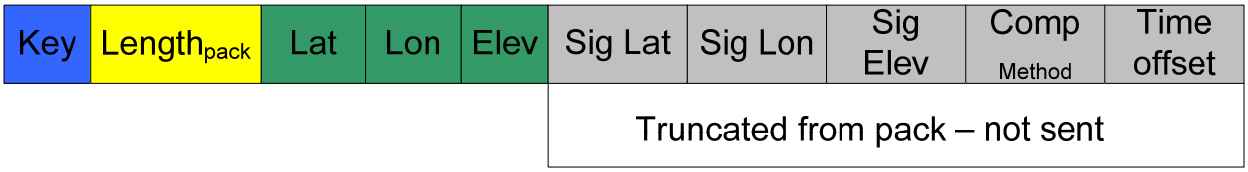
A truncated pack is a normal KLV fixed length pack that can have has its length value set so that only a portion of the values of the pack are used. For example the following diagram shows 4 values in a pack; the normal length of this pack would be: 8+12+8+16 = 44 bytes.



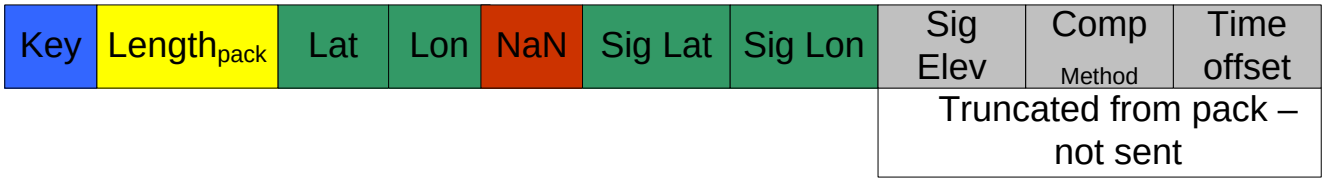
If an application decides that Values 3 and 4 are not going to be computed or sent in the KLV then a truncated pack can be used just by setting the length to: 8+12 = 20 bytes; this is shown in the figure below.



There are two issues with using truncated packs, element order is important (high priority items first) and the applications using them must be sure to truncate complete values and not partial values. If a truncation results in a partial value then the decoder must skip over decoding the partial value. When the pack is designed, the order of the values has to be defined so the most important items come first – if truncation occurs, then the low priority items are removed from the pack. An example of a truncation pack order would be a pack that contains latitude, longitude, elevation, latitude accuracy, longitude accuracy, elevation accuracy, computation method and Time offset; by putting the latitude, longitude and elevation items first the rest of the pack can be truncated – *i.e.* removing latitude accuracy, longitude accuracy, height accuracy, computation method and Time offset from the pack – leaving the most important items (the latitude, longitude and elevation). With known prioritization of values in a pack bandwidth issues can be easily addressed by just changing the value in Length$_{pack}$. The following diagram illustrates the above example.



All values in the pack must be supplied, values cannot be skipped or dropped in packs prior to truncating the pack. Sometimes it is necessary to supply filler values when a pack value is unknown. **Each value listed in the RP 0702 companion document must define a filler value**; an example is to use IEEE NaN for floating point values. Changing the preceding example slightly, the lat/lon, sig lat and sig lon are known values but the elevation value is unknown so it must use a filler value of NaN as shown below.

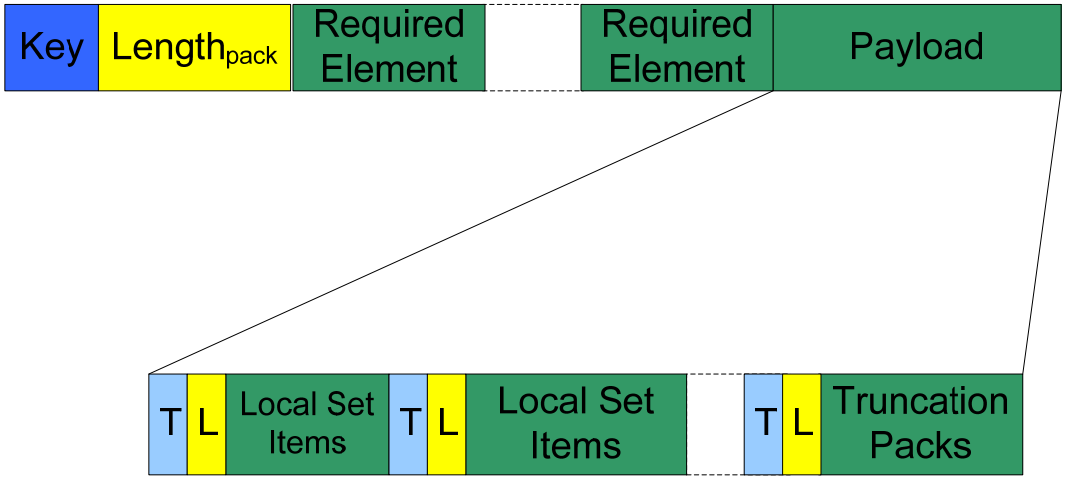


4.2.4 Combining Floating Length and Truncation Packs

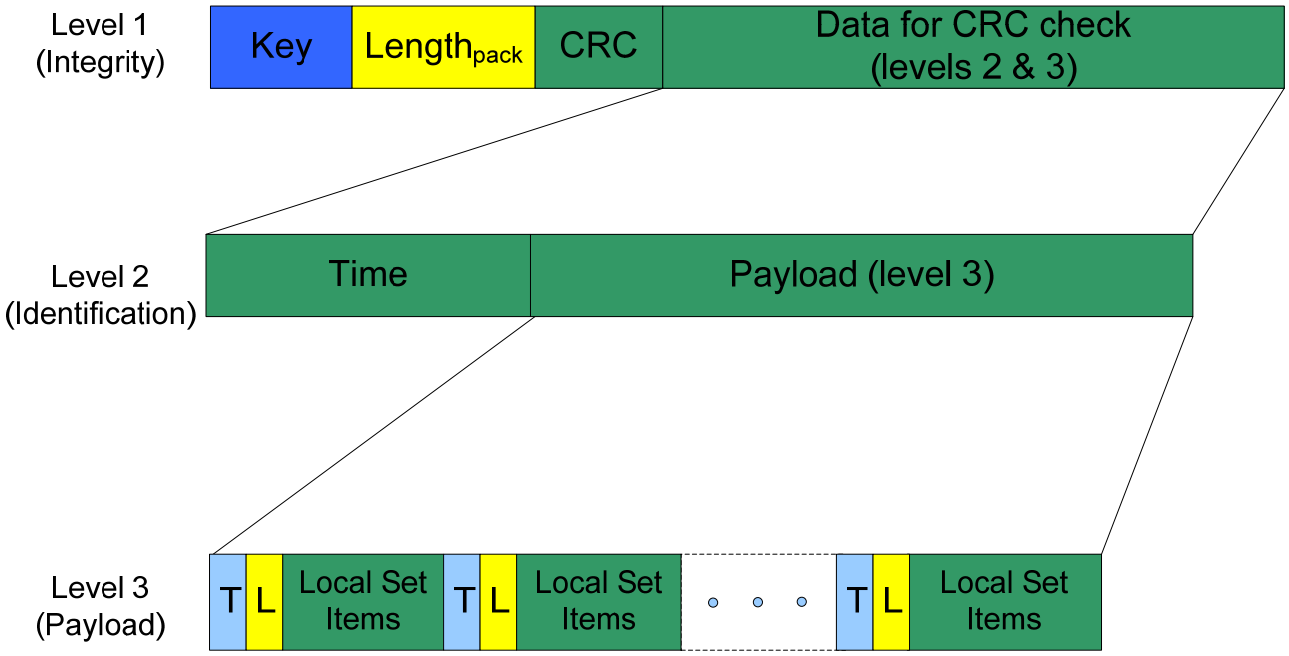
Both floating length and truncation packs concepts can be combined into one pack however the usefulness of the resulting pack may be limited. Both pack structures use the length to determine how to “use” the pack. Since the floating length pack’s only variable length element is the last element and the truncation pack removes the last element(s) as the length shrinks then combining both really does not provide too much capability. It is not recommended to combine these pack structures.

4.3 CMS Packet Structure

The CMS packet structure is comprised of several embedded KLV structures. The top KLV structure is a Floating Length Pack (defined in Section 4.2.2) which contains the CMS Key, CMS packet length and a list of fixed length required elements followed by the final variable length element. The final variable length element is a Local Data Set that contains Local Data Set items, each of which can be either single elements, Truncated packs or other KLV constructs (*e.g.* Universal Sets). The following diagram illustrates the CMS structure:



A CMS packet is organized into a hierarchy with three levels: Integrity, Identification and Payload. The purpose of the Integrity level (level 1) is to insure that all data in the CMS packet is valid. The purpose of the Identification level (level 2) is to insure that every CMS packet has the time of the packet. The purpose of the Payload level (level 3) is to contain all other sensor/platform and administrative metadata. The following diagram illustrates a complete CMS packet:



The integrity level contains two items, a CRC and the CRC data to validate. For the base CMS error detection system the CRC value is a two byte value that is the result of the CRC-16-CCITT ($x^{16} + x^{12} + x^5 + 1$) algorithm executed over the CRC data to validate – see Appendix C- CRC Algorithm. The CRC data to validate is the Identification and Payload data (*i.e.* levels 2 and 3).

The Identification level contains two items, a CMS packet time value and the Payload data (i.e. level 3). For the CMS base timing system the packet time value is the POSIX based time

value, which is an 8 byte count of the number of microseconds since 12 midnight January 1 1970, based on RP 0603. The time value must always be an increasing value for each packet; **it is important that the time value never decreases**. The time value is intended to be the time that the data in the payload was measured – see Section 4.3.3 - CMS Timing Rules, below. To increase the reported timing accuracy of when data measurements occurred each measured value in the payload can have a time offset (relative to packet time) associated with it – see Section 4.3.2.1.

The Payload level is the final item of the CMS top level floating length pack, it is a SMPTE 336M local data set **value** (key and length are not needed since this is an item within the Pack syntax). The value is composed of a series of items encoded using local set BER OID Tags and BER lengths. The value contains the sensor/platform metadata either individually or in truncated packs. For individual items in the CMS payload the tag, length and value are called Taglets. For truncation packs in the CMS payload the tag, length and all of the values (for the truncation pack) are called Packlets. The definition of the payload item tags, lengths and values are in a companion RP 0702 Content Document. Truncation packs are used to group related items and reduce the number of bytes needed to transmit the data.

4.3.1 Data Frequency Rules

There are two categories of payload data items: static and dynamic. Static data items are data that do not change over the life of the stream (*e.g.* Mission). Dynamic data items change at some rate. Since downstream applications need to process this data starting at any point in the KLV stream the static information must be available within a known time interval within the stream. The maximum time between static packets shall be 60 seconds. Injecting the static data into the stream at faster rates, however, is acceptable and, depending on bandwidth, is encouraged.

In addition to static data being available every 60 seconds all of the dynamic data should also be recovered within a 60 second window; this is called history data. In order to denote that the data being received by a consumer is historical data (and not currently measured) the historical data shall be sent in CMS packets but with a different CMS key – see Section 4.1.1. For example, a piece of dynamic data may only change once per hour, however it needs to be reported in the stream at least once every 60 seconds. For downstream applications it is a simple task of monitoring the stream for 60 seconds looking for historical CMS packets to obtain all of the metadata that a system is producing. The data received in a historical packet is assumed to not have a measurement time – the consumer of the data has no knowledge when the measurement occurred.

The payload does not have elements that are required in every packet; however, there are two static data items required in the history packets: a unique identifier for the stream and processing information about the stream. The unique identifier for the stream, called a Universally Unique Identifier or UUID, is generated using a standard algorithm (see RFC 4122) and provides a method of identifying the stream for down stream processing. The UUID usage is discussed further in Appendix B. The processing information contains three components which identify what system produced the data, the version number of the system and the standards (this standard) version number (RP0701.0). See Appendix A for more information on Stream Processing information.

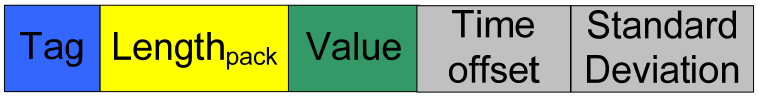
4.3.2 Payload Items

Within the payload level, payload items are inserted based on the tags and definitions in RP 0702. To provide the most flexibility individual items (*e.g.* latitude, longitude, *etc.*) can be placed into the payload – each individual item will require a tag, length and value. The tag, length and meaning of each of these items are defined in the RP 0702 document. To improve efficiency (truncation) packs can be formed which contain multiple items; each of these packs will also be defined in RP 0702. Since one of the goals of CMS is to provide a place in the payload for values to have their measurement uncertainty defined each value would need to have an uncertainty measurement identified in RP 0702. Additionally, each value can have its own time offset (described in section below) which is also defined in RP 0702. To simplify the packet data, reduce bandwidth and reduce the number of definitions in RP 0702, truncation packs are used to combine the value, measurement uncertainty, and time offset into one local set item. This means that for each fixed length value in RP 0702 the standard deviation and time offset are automatically included within the definition of the value. This value is then called an extended value, since its definition is extended to include more than just the value.

In RP 0702 there are three types of payload items, **single value**, **extendable value** and **pack**.

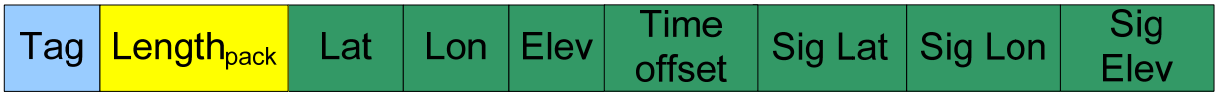
The **single value type** means that there is only one element in the payload item – usually for variable length items.

The **extendable value type** means that there is an element with some additional attribute data further refining the meaning of the element. If the value in the payload item is fixed length then it can be extended with other information and treated as a truncation pack. Possible extended values are: time offset and standard deviation. The following shows an extended value truncation pack:



The purpose of the extendable type is to reduce the number of tags in the payload; by allowing each fixed length value to have default attributes (time offset and standard deviation).

The **pack type** means that the “value” is a collection of other single values in a truncation pack. The following illustrates a pack type which is the combination of several individual values.



Pack types are defined by their “tag” and the definitions of the packs are described in the RP 0702 companion document.

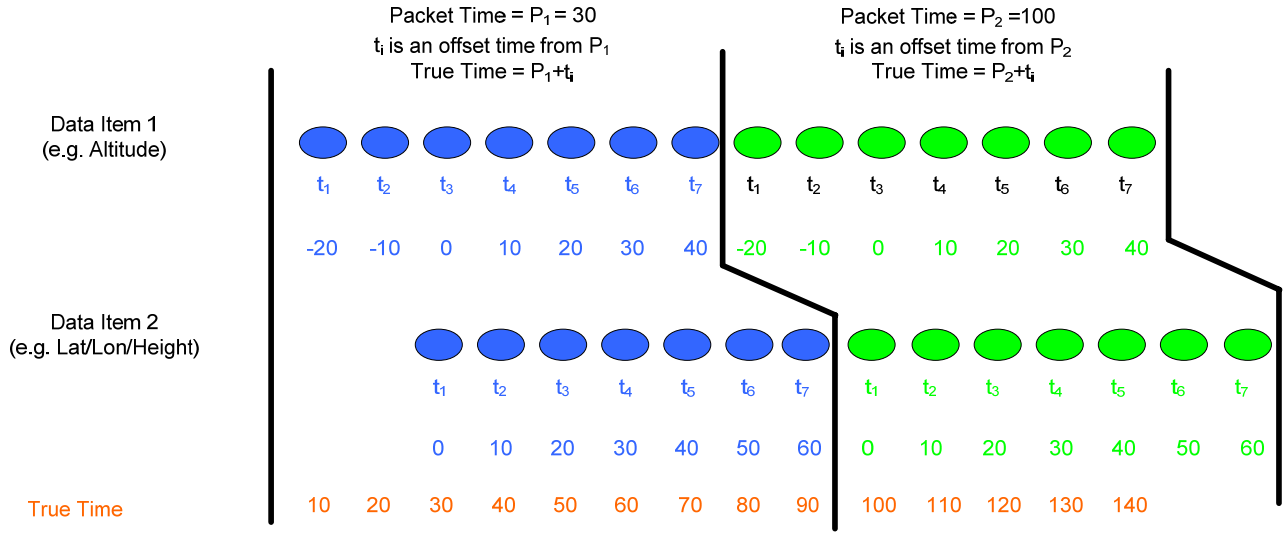
Each payload item can be included multiple times in the payload – for the extended and pack value types, this allows for series of the same measurement(s) recorded at different times.

Care must be taken by the KLV encoder to insure that the values do not overlap temporally between packet times – that is discussed below.

4.3.2.1 Time offset

Each extended value or pack type can have a time offset(s) assigned to it. The time offset is an additive or subtractive number of microseconds from the packet time. When the time offset is added to the packet time the absolute or true time is formed for that measurement. The true time value must always be an increasing value for each measurement. Since motion imagery is defined at a rate of at least one frame per second the maximum size of this offset value is ±1,000,000 microseconds which, when rounded up to the nearest byte size, takes 24 bits or 3 bytes. This offset value shall never exceed ±1,000,000 microseconds.

The time offsets for each extended or pack item is independent of all of the others. This provides the most flexibility for system developers when assigning time values. The following illustrates a timing example. The blue items show all of the values of two different measurements (altitude and geo-position) for one "packet" of data; the green values show similar measurements for the following packet. This diagram shows that the values for altitude and geo-position do not have to be temporally linked – they are independent of each other.



The above illustration shows blue ovals for data items each with their own time offsets t_i inserted into a CMS packet. The packet time is 30 microseconds and the individual offsets are shown in blue text below the ovals. The bottom line of numbers is the true time that is computed from the Packet Time and time offsets. Likewise the true times are shown for the next CMS packet (shown in green) with a packet time of 100. This illustration shows that all of the data items across both packets have monotonically increasing time values.

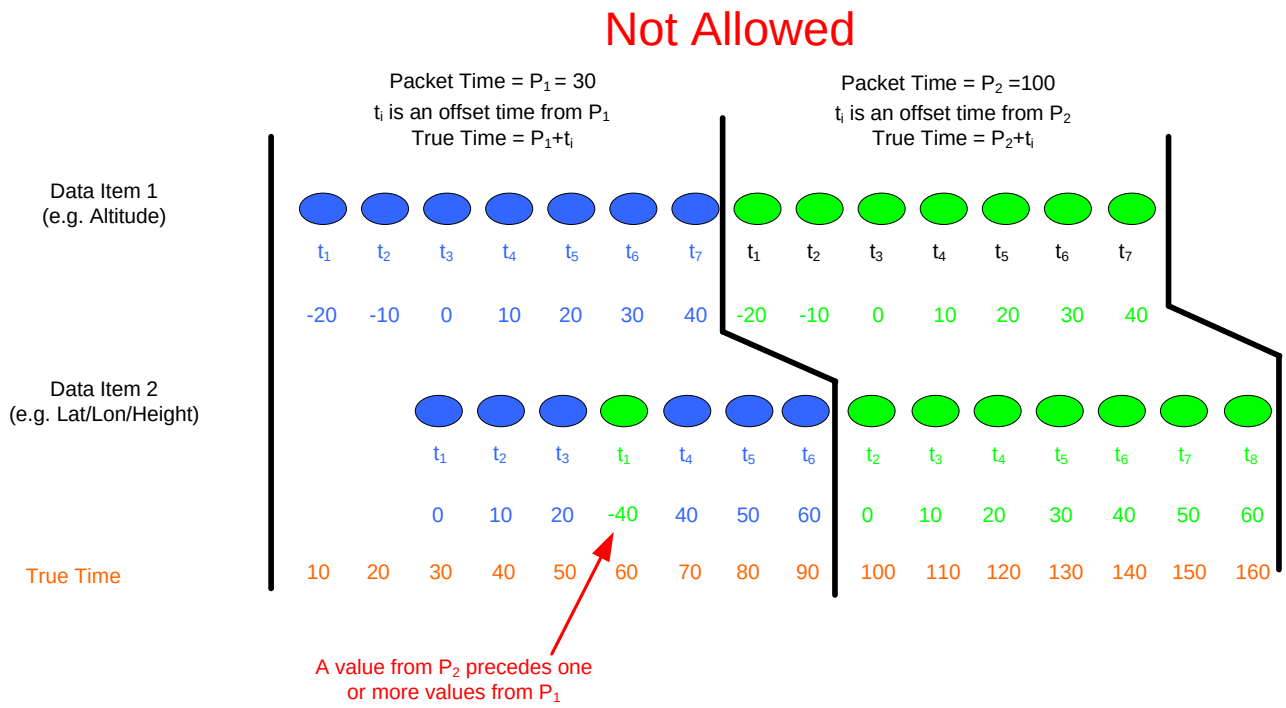
There is a potential problem with offset times when used for different instances of the same data item across two consecutive CMS packets. It is possible to have the data item's time offset in one packet temporally precede other instances of the same data item in a previous packet – this problem is called temporal pack distortion.

The CMS KLV encoder shall ensure that the KLV times for each data item are not temporally misplaced in time with instances in a different packet; KLV encoders shall prevent temporal pack distortion. For example:

> *If packet 1 has a timestamp value of 1,155,042,000,000,000 (2006-08-08 13:00:00.000) and has a slant range value in it (with time offset 0); this means the true time for the slant range measurement is 2006-08-08 13:00:00.000. Packet 2 has a time value of 1,155,042,000,100,000 (2006-08-08 13:00:00.100) and has a slant range value in it with offset -500,000 (1/2 second); so the true time value is 2006-08-08 12:59:59.600. This means that the slant range in packet 2 came before the slant range in packet 1 – this is illegal!*

The following shows an illustration of the offset timing within a packet and the temporal packet distortion issue described above.

The following example shows a data item (P_2, t_1) from packet 2 temporally precedes data items (P_1, t_4), (P_1, t_5) , (P_1, t_6) from packet one, resulting in a temporal pack distortion; meaning that the data items true times are not monotonically increasing over both packs.



The above example shows an example of a temporal pack distortion that CMS KLV encoders must prevent from happening.

If a decoder detects temporal pack distortion then all values in a successive CMS packet (*e.g.* P_2) that are temporally marked before values in the preceding CMS packet should be ignored.

4.3.3 CMS Timing Rules

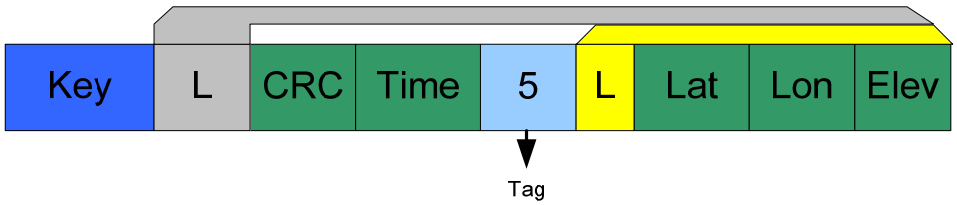
CMS provides a lot of flexibility for indicating the time of any measured value using the combination of the CMS packet time and the time offsets described above. For doing precise geo-registration and precision targeting it is important to have accurate metadata measurements synched with the video timing. In order to provide all information with as little latency as possible it may not be possible to synchronize all of the data before transmitting it. To alleviate this problem the platform/sensor vendors (*i.e.* KLV producers) can timestamp all metadata being measured; this allows later processing to correlate temporally the metadata measurements with the video to attain the best possible accuracy.

To allow KLV producers to indicate their timing capabilities the uncertainty of the timing can be specified for the whole packet, with a taglet described in the CMS – Content document (RP 0702). By not including the uncertainty taglet in the CMS packet indicates that the timing uncertainty is unknown. Future versions of this standard will allow the timing uncertainty of individual items to be specified.

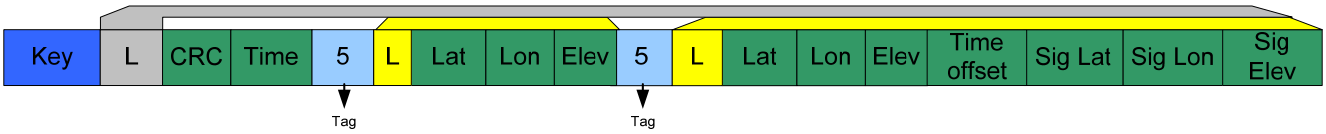
4.4 Metadata updating and additions

During downstream processing (*e.g.* exploitation, coordinate correction, *etc.*) certain function may need to augment or correct values that are in the CMS KLV data packets. It is important to note that when possible the original data should not be destroyed, instead the new KLV should be appended to the CMS packet or new CMS packets should be created. For numeric values that have been corrected or improved the associated standard deviation should reflect an improvement over the original value. For example: If a latitude/longitude value is improved then the new measurement uncertainty should be added so that future users know which value is more accurate – the original (which may or may not have a measurement uncertainty) or the new improved value.

The following diagram illustrates a CMS packet with one item in it identified with Tag Id 5. This fictitious item is a truncation pack with the value portion of the item contains a latitude, longitude and elevation; the remaining parts of the truncation pack are time offset, sigma lat, sigma long and sigma elevation.



During later processing the latitude, longitude, elevation values could be updated along with adding the measurement uncertainty values. Instead of replacing the original values in the CMS packet the new and improved values should be added on to the existing CMS packet – leaving the original data untouched. Note that the length and CRC values for the CMS packet would have to be recomputed. Additionally the time offset would be set accordingly or filled in with a filler value.

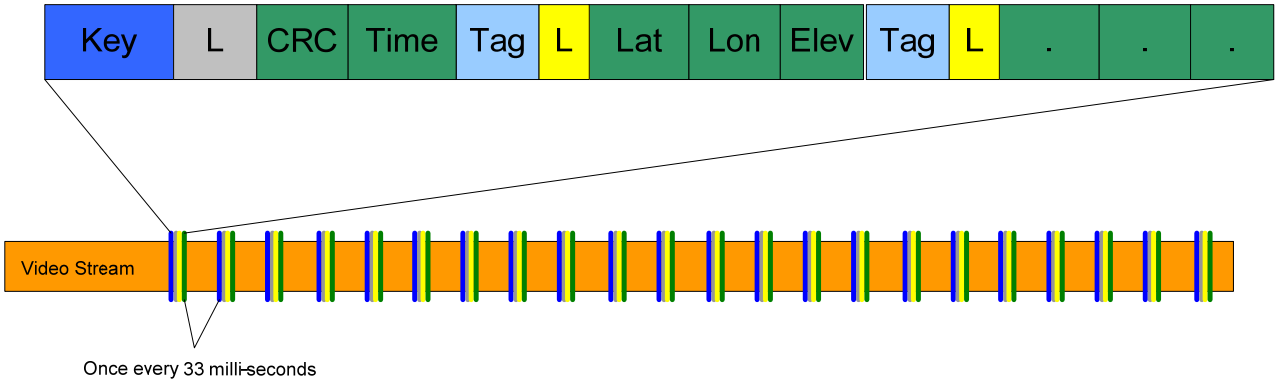


The one exception to this rule is when the classification value of the CMS packets or stream needs to be changed. For this case the old classification would be removed and the new classification will be added. See RP 0102 for information on security structure and content.

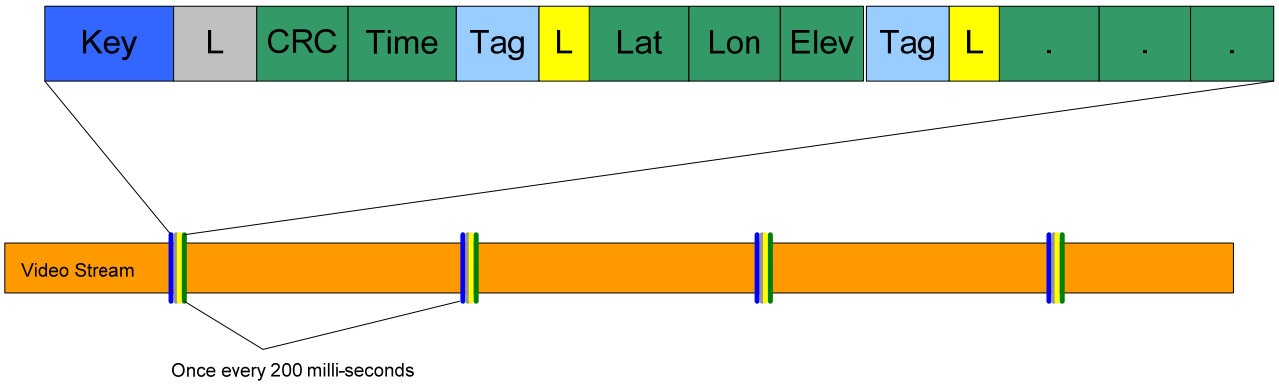
4.5 Repackaging KLV for lower bit rate distribution (Informative)

For low bandwidth distribution there are times when the metadata needs to be reduced to preserve bits. There are two choices when reducing the bandwidth, one choice is to reduce the frequency rate at which each packet is sent; the other choice is to reduce the size of each packet. Since truncation packs have been employed in this standard, bandwidth reductions can be quickly performed by further truncating values from the packs.

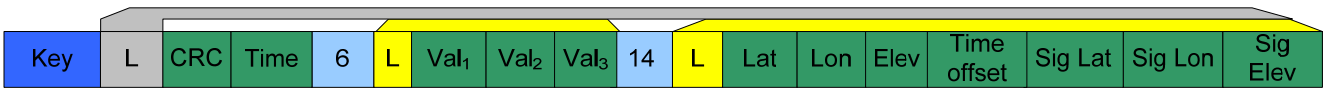
The following illustrated example shows a CMS packet being sent once every 33 milliseconds:



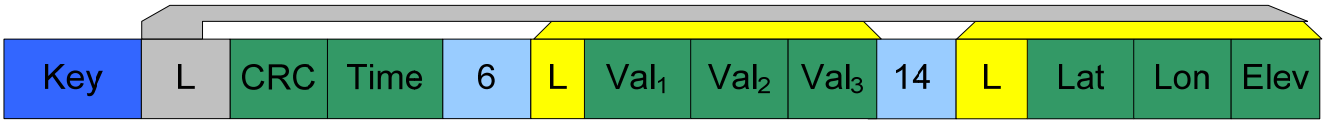
The following illustrated example shows the bandwidth being reduced by reprocessing the stream and limiting the number of packets sent to once every 200 milliseconds:



The following is an illustrated example of a CMS packet with a truncation pack embedded in it:



To reduce the bandwidth the “less-important” values can be truncated from the pack reducing the bandwidth as shown in the following illustration.



Both of these techniques can be employed at the same time reducing the bandwidth required to send the essential data to the end user.

5 Informative Examples

NA

Appendix A - Processing Version Information

The purpose of the processing information is to identify who, what and how the KLV data was generated. This identification data is used to insure that the correct decoders and definitions are being used when processing the CMS packets. The processing information contains the following:

- RP 0701 version,
- RP 0702 version,
- Producer name and Producer Version number.

The RP 0701 version is the version number of this RP that has been implemented as a single byte (e.g. 0701.11 would use “11” as the value). The RP 0702 version is the version of the content document as a single byte. The Producer name is the name of the software and/or company that was used to generate the metadata and the Producer Version number is the version number of the software used to generate the metadata. This value should be formatted as a string, separated by colon. For example: “ACME GOV SYS:5.3”. These three values can be easily combined into a pack; which is defined in RP0702.

Appendix B - Universally Unique Identifier (UUID)

The UUID is to be used as a way of identifying each individual stream from a sensor/platform. The UUID is a 128 bit value that can be generated easily within a platform or ground control station. Once the UUID is created it is not changed for the entire mission; the 128 bit value put into the CMS packet is the same for all packets during the mission. This allows multiple users to identify that they are working on the same data.

Today, nominal missions are around 30 hours so the UUID can be used to easily identify a mission. Future systems may have much longer durations – weeks or months at a time. In this case the UUID can be used in conjunction with the time value to identify any segment of the mission of interest.

Please refer to RFC4122 for more information on the UUID construction and generation algorithm. In this RFC there is a c-code example for UUID generation. Additionally, Java 1.5 and beyond has a UUID class that can be used to generate UUIDs.

Appendix C - CRC Algorithm

The CRC algorithm is used to detect corruptions when transmitting data from one point to another. The algorithm chosen for CMS is CRC-16-CCITT. The following is an example implementation in Java. This example contains some test values for validation.

```
public class CRC_16_CCITT {
  public final static int[] table = createTable(4129, 16, 8);
  /**
  ************************************************************************
  * createTable
  * Makes the CRC table
  *
  * @param poly
  * @param width
  * @param lookUpSize
```

```
* @return
*******************************************************************************
*/
private static int[] createTable(int poly, int width, int lookUpSize) {
  int size = (int)Math.pow(2, lookUpSize);
  int[] table = new int[size];
  for(int i = 0; i < size; i++) {
    table[i] = tableElement(i, poly, width, lookUpSize);
  }
  return table;
}

/**
*******************************************************************************
* tableElement
* Makes the table element
*
* @param ival
* @param poly
* @param width
* @param lookUpSize
* @return
*******************************************************************************
*/
private static int tableElement(int ival, int poly, int width, int lookUpSize) {
  int topBit = (int)Math.pow(2, (width - 1));
  int index = ival * ((int)Math.pow(2, lookUpSize));
  int mask = (int)Math.pow(2, width) - 1;
  for(int i = 0; i < lookUpSize; i++) {
    if((topBit & index) != 0) {
      index = (index * 2) ^ poly;
    } else {
      index *= 2;
    }
    index &= mask;
    // System.out.println(topBit + ", " + index + ", " + mask);
  }
  return index;
}
/**
*******************************************************************************
* getCRC
* Gets the CRC value for this byte array
*
* @param key the byte array
* @return the CRC
* @throws IllegalStateException if error
*******************************************************************************
*/
```

```
  public static int getCRC(byte[] key) {
    // poly = x^16+x^12+x^5+1
    int tVal = table[255];
    int aVal = (tVal >> 8);
    int lastBVal = (tVal & 255);
    int lookUp = (aVal ^ 255);
    int bVal;
    for(int i = 0; i < key.length; i++) {
      tVal = table[lookUp];
      aVal = (tVal >> 8);
      bVal = (tVal & 255);
      lookUp = aVal ^ lastBVal ^ key[i];
      lastBVal = bVal;
    }
    tVal = table[lookUp];
    aVal = (tVal >> 8);
    bVal = (tVal & 255);
    lookUp = (aVal ^ lastBVal);
    int returnVal = ((lookUp * 256) ^ bVal);
    return returnVal;
  }
  /**
  ********************************************************************************
  * printVal
  ********************************************************************************
  */
  public static String strVal(int val) {
    return  val + " ("+ Integer.toHexString(val).toUpperCase() + ")";
  }
  /**
  ********************************************************************************
  * main
  ********************************************************************************
  */
  public static void main(String argv[]) {
    System.out.println("CRC Tests\n table:");
    //for(int x = 0; x < (table.length - 1); x++) {
    //  System.out.print(table[x] + ",");
    //}
    System.out.println(table[table.length - 1]);
    System.out.println("Table.length=" + table.length);
    String s = "";
    for(int x = 0; x < 256; x++) {
      s += "A";
    }
    System.out.println(" test 1 --> " + strVal(getCRC(s.getBytes())) +" --> 256 A's, should be 59704
(0xE938) ");
    byte[] shorty = {3, 5, 11};
    System.out.println(" test 2 --> " + strVal(getCRC(shorty)) + " --> 3 5 11, should be 1730");
```

```
    byte[] shorty2 = {65};
    System.out.println(" test 3 --> " + strVal(getCRC(shorty2)) + " --> 65, should be 38009 (0x9479)");
    s = "123456789";
    System.out.println(" test 4 --> " + strVal(getCRC(s.getBytes())) + " --> chars 1 to 9, should be
58828 (0xE5CC)");
  }
}
```