# THE IRIG 106 CHAPTER 10 SOLID-STATE ON-BOARD RECORDER STANDARD: A DATA PROCESSING PERSPECTIVE

**Tim Thomas**
**TYBRIN Corporation**
**96th Communications Group**
**Test and Analysis Division**
**Eglin AFB, Florida**

## ABSTRACT

The Telemetry Group (TG) of the Range Commanders Council (RCC) developed the Chapter 10 addition to the IRIG 106 standard to "establish a common interface standard for the implementation of solid-state digital data acquisition and on-board recording systems" [1]. This standard is intended to allow the development of a common set of data playback/reduction software, minimizing the need for a large number of unique programs to handle proprietary data structures. This paper analyzes the Chapter 10 standard from a data processing perspective, providing insight into the benefits and challenges developers will face when writing Chapter 10 software.

## KEY WORDS

RCC, IRIG 106 Chapter 10, software implementation, digital data recorder

## INTRODUCTION

With the introduction of the IRIG 106 Chapter 10 standard, the 96th Communications Group (96th CG) at Eglin AFB, Florida, has taken the opportunity to begin the development and maintenance of a set of software applications to process Chapter 10 compliant data. The main focus of the 96th CG software is to perform data reduction and analysis processing on Chapter 10 data that has been downloaded from a solid state recorder (SSR) to a PC-readable disk file. From this experience, the software engineers of the 96th CG have gained valuable insight into both the benefits and challenges other developers may encounter in the development of software to handle Chapter 10 compliant data.

# CHAPTER 10 OVERVIEW

## Data Organization

A Chapter 10 file stores data from multiple data streams and is organized into packets as shown in Figure 1. The first packet of the file must be a Computer Generated Data Packet, Format 1 Setup Record. The content of this packet follows the IRIG 106 Chapter 9 Telemetry Attributes Transfer Standard (TMATS). The second packet in a Chapter 10 file must be a time data packet. This packet is then followed by one or more data packets, including subsequent time channel packets, if any.


**Figure 1 Chapter 10 Data Organization**

## Packet Format

Each packet of a Chapter 10 file follows the general packet format shown in Figure 2. Every Chapter 10 packet begins with a common fixed length packet header optionally followed by a secondary packet header. The packet body format is dependent upon the data type of the packet. Packet sizes can be as large as 512 KB (524,288 bytes) except the first packet in the file which can be as large as 128 MB (134,217,728 bytes).

| | |
|---|---|
| Packet Header | Packet Sync Pattern |
| | Channel ID |
| | Packet Length |
| | Data Length |
| | Header Version |
| | Sequence Number |
| | Packet Flags |
| | Data Type |
| | Relative Time Counter |
| | Header Checksum |
| Secondary Packet Header (optional) | Time |
| Packet Body | Channel Specific Data |
| | Data 1 |
| | Data 2 |
| | … |
| | Data N |
| Packet Trailer | Filler (optional) |
| | Data Checksum (optional) |

**Figure 2 Chapter 10 General Packet Format**

## BENEFITS

Writing data reduction and analysis software has never been an easy task for the software engineer. Fortunately, the Chapter 10 standard provides many benefits that simplify the software writing task.

### Easier to Learn

Before writing a data processing application the software developer must negotiate the learning curve of various data standards. Handling different proprietary recorder formats and decoding a variety of cryptic Interface Control Documents (ICD) are among many of the difficulties that we often have to face. The advent of the Chapter 10 standard and its increasing acceptance among vendors brings the promise of a much easier learning curve to climb.

The Chapter 10 standard is by no means an easy standard to learn. Although the basics are fairly straightforward, having to learn one standard one time is much better than having to learn a completely different standard for each proprietary format. The document is well-organized and well-written, unlike many ICDs that we have dealt with before. Chapter 10 may be voluminous but that is due to good coverage of the many details of the standard.

### Readily Available

Another difficulty with processing proprietary formats is obtaining the necessary documentation. Because vendors tend to treat their proprietary information as company confidential, such documentation is usually not publicly available. To obtain such information often involves a negotiation where the requester must provide a legitimate need for the information and agree to nondisclosure of the information. This also makes it difficult to keep up with changes in the proprietary standard as there often is not a feedback mechanism in place to notify the requester of changes to the standard.

Fortunately, Chapter 10 does not have these problems. It is readily available for download from the RCC site and updates are published regularly.

### Easier Decoding

Processing data extracted from a file can generally be divided into two tasks: decoding the format and decoding the data. Decoding the format involves identifying and locating a data channel within the data record and extracting the time stamp and raw data. Decoding the data involves processing the raw data into meaningful numbers (e.g., decommutation and engineering unit conversion of PCM data). Chapter 10 simplifies the task of decoding the format.

Simple Packet Format – The Chapter 10 packet format is conceptually very simple. It is composed of a fixed-format header that is common to all Chapter 10 supported data types and a variable-sized packet body that follows a well-ordered generalized format. The channel data is thus located at a well-known location in the packet body and the

straightforward data layout (i.e., intra-packet time, intra-packet header, data words, etc.) facilitates extraction.

One Channel per Packet – A Chapter 10 packet eliminates the need to determine what channels are contained in the packet and where in the packet they are located. The data type and channel are identified in the packet header and the packet body contains data from that specific channel only.

Variable Packet Size – Squeezing data into fixed-size formats is a tricky business. Depending on the data type, there may be unused space at the end of the record or logically contiguous data (e.g., PCM frames and 1553 messages) must be split across multiple records. Chapter 10 solves this problem by providing for variable packet sizes to fit the specific requirements of each data channel. One caveat, however, is that the standard does not prohibit the splitting of data across packets. It merely provides a way to avoid doing so.

**Easier Maintenance**

An obvious benefit of standardizing a recording format is the smaller number of software applications that need to be developed. This of course helps to economize on software development resources. But a less obvious benefit comes from the design of the general packet format. As mentioned previously, the Chapter 10 packet header is a fixed-format that is common to all Chapter 10 supported data types. With this kind of packet structure, each packet in a Chapter 10 file can be processed in a generic manner without having prior knowledge about the type of data in the packet body.

This support for generic processing allows the developer to apply the key software engineering principle of separation of concerns. It is very easy with the Chapter 10 packet format to separate the code that processes the packet header from the code that processes the packet body. By keeping packet header processing functionality (e.g., building a file index, compiling file statistics, or browsing packets) separate from packet body processing an application can be made robust in the presence of new data types.

Of course, data-specific handling of the packet body will require additional code each time a new data format is added to Chapter 10. Fortunately, the packet structure can help simplify this task by supporting a polymorphic approach to handling new data types. Rather than having to modify---or even rewrite---existing code, the software engineer can architect the application to handle a new data type with the addition of a new implementation class, thus leaving existing code undisturbed.

## CHALLENGES

No data standard is ever perfect. As powerful as the Chapter 10 standard is in easing the burden of the software engineer, there are a number of difficulties that will likely be faced when developing a Chapter 10 software application.

**Random Access**

It is common for data formats to be comprised of fixed-length blocks, or records, as shown in Figure 3. Traversing through such a data file is a fairly simple matter of repeatedly reading in the same number of bytes. Random access of records in such a file is also very straightforward. The file offset of any record can be calculated with the following formula:

$$P = R(N - 1)$$

P is the file offset of the desired record
R is the record size
N is the desired record number

Using this file offset, the desired record can now be read. This operation can be performed with O(1) complexity, thus accessing any desired record in the file can be done in constant time.


**Figure 3 Generic fixed-length record format**

With Chapter 10 data files, however, it is a different matter. Packets in a Chapter 10 file can be of different lengths so we cannot traverse a file by simply reading in fixed-length records. The TG took care of this problem by including the packet length as a field in the header section of the packet. Each packet in a Chapter 10 file thus provides a link to the following packet, as shown in Figure 4. This in effect makes a Chapter 10 data file equivalent to a data structure known as a singly-linked list.

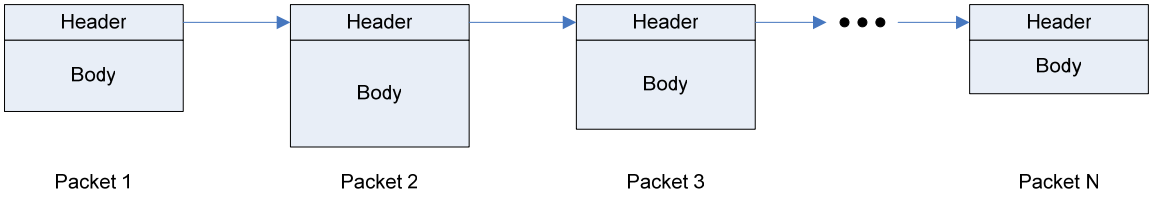


**Figure 4 Chapter 10 data file as a singly-linked list**

Constant time access of packets in this structure is not possible. To find a particular packet requires a sequential search through the file. The computational complexity for retrieving a particular packet N from a singly-linked list is O(N). Thus, the time required to search for and retrieve a particular data packet increases linearly for each packet in the file. For relatively small data files, and for applications where it is unnecessary to search for particular packets, this poses little problem. However, when dealing with applications

that require processing on a subset of packets within a larger file (e.g., processing a specific data channel or within a certain time interval), the application running time can become quite prohibitive.

Version 2.1 of the Chapter 10 standard appears to address this issue by defining an index packet for direct access to packets in the Chapter 10 file. However, this indexing scheme does not provide a true random access capability.

**Resolving Time**

Processing time data is usually a straightforward operation. Time stamps are typically provided as an absolute time of day (TOD), such as IRIG-B, and are relatively easy to process. This makes time accuracy more of an upstream (hardware engineer) and downstream (data analyst) problem. The job of the software engineer—usually—is simply to extract and use the given time.

Chapter 10, however, adds a wrinkle. It allows time to be represented as either a 64-bit TOD value or a 48-bit relative time counter (RTC) value that comes from a free-running 10 MHz binary counter on board the recorder. If the RTC value is used the software will need to use packets from the time channel in order to resolve the RTC value into TOD. This puts the time accuracy problem into the domain of the software engineer because now the software must determine the TOD from other data in the file.

Theoretically, determining the TOD from the RTC is a simple matter of linear extrapolation using a time channel packet. This can be done with the following formula:

$$T_D = T_T + \frac{R_D - R_T}{10{,}000{,}000}$$

$T_D$ is the TOD (in seconds) of the data
$T_T$ is the TOD (in seconds) from the time channel packet
$R_D$ is the RTC time stamp of the data
$R_T$ is the RTC from the time channel packet

This should work with any time packet from the time channel. But this only works by assuming that every 10,000,000 ticks of the RTC equate to exactly one second intervals from the time source. If the RTC values do not monotonically increase by 10,000,000 counts from one time channel packet to the next (assuming a one second interval time source) then using different time packets will produce different TOD values. In one particular file that we examined, for example, the difference between using the first packet and the last packet of the time channel produced TOD deviations as high as 116 ms.

The obvious approach to addressing this issue—assuming the recorder RTC is the problem and not the time code generator—is to use interpolation between two time packets instead of extrapolation from one (if the time code generator is the problem, no

software solution suggests itself). This approach is also fairly simple to implement with the following, slightly more complicated, formula:

$$T_D = T_1 + (T_2 - T_1)\frac{R_D - R_1}{R_2 - R_1}$$

$T_D$ is the absolute time (in seconds) of the data
$T_1$ is the TOD (in seconds) of the first time packet
$T_2$ is the TOD (in seconds) of the second time packet
$R_D$ is the RTC time stamp from the data
$R_1$ is the RTC value from the first time packet
$R_2$ is the RTC value from the second time packet

Such a first-order solution would help to significantly reduce time processing errors due to such problems as oscillator crystal jitter and drift. Unfortunately, it adds another layer of complexity to the processing. Not only does the software have to locate and extract the desired packet, but now it must also locate and extract the time channel packets immediately preceding and following the data packet. Higher order solutions using multiple time packets, such as polynomial interpolation or adaptive filtering techniques, might be considered but it is doubtful the additional processing overhead would measurably improve results.

**Format Stability**

Data standards rarely remain fixed after their initial release. They are continuously changing as they evolve and improve to meet new needs and changing situations. As a data standard changes during its lifetime, this inevitably leads to changes in format definitions. Each format change adds complexity to software written to that standard. Chapter 10 is no different.

We have already seen this type of change in the Chapter 10 standard with the addition of the Packet Secondary Header option. While it is impossible to anticipate all future changes, it is highly desirable in the early stages of the development of a standard to identify those areas where future changes are likely to occur. By designing the standard to be as accommodating as possible to future changes the need for further software changes can be reduced. An analysis of the Chapter 10 general packet format reveals some weaknesses in this area.

Data Type – An 8-bit field is provided in the packet header to specify the particular data type of the packet. This allows for defining up to 256 different data types. While this may seem adequate, experience has shown that the number of available data types is continuously growing. The Chapter 10 standard already specifies, or reserves, 104 of these 256 values, leaving 152 still available. It seems probable that these remaining values will be used up in the near future.

Packet Flags – The packet header provides for eight bit flags. All eight of these flags have been used and it is a virtual certainty that new flags will need to be defined in future upgrades. This field is definitely not large enough to accommodate future changes.

**Sequence Number**

Sequence numbers are a good way to check for missing data. The packet header sequence number is an 8-bit number that will roll over every 256 packets in a data channel. While this field is useful to some extent, the small period will result in sequence numbers being duplicated many times in channels with large numbers of data packets. This would allow packets to be mixed up during processing or large numbers of packets to be lost without being easily detected. If a gap in sequence number was detected, it would be difficult to know for certain how many packets were actually lost. It would be more useful if the sequence number field was large enough to greatly reduce these risks by ensuring that each packet in a data channel has a unique sequence number.

**Checksums**

Checksums are a commonly used means for error detection. Chapter 10 specifies a 16-bit additive checksum for the packet header and the choice of an 8-, 16-, or 32-bit additive checksum, or none at all, in the packet trailer. The reliability of a checksum depends on its size—larger is usually better—and the algorithm used to compute it.

8-bit checksums are very weak. They provide a relatively high probability (1/256) of failing to detect a randomly induced error. 16- and 32-bit checksums are stronger, providing lower probabilities (1/65536 and 1/4294967296 respectively) of failing to detect a randomly induced error.

Additive checksums are very simple but also very weak. They fail to detect errors such as transposed bytes or multiple errors that cancel each other. Other algorithms such as Cyclic Redundancy Check (CRC) algorithms are much better for detecting these and other kinds of errors. Cryptographically strong hashing algorithms are even better but they are computationally intensive and are best left for applications that require detection of maliciously induced errors.

**Secondary Packet Header**

The optional secondary packet header presents an interesting set of problems. The first problem is the fact that it is optional. The software has to perform extra processing steps to detect and if necessary extract the secondary packet header. The second is what to do with the secondary time. Although the time from the secondary packet header is in an easier to use format, it is redundant.

## RECOMMENDATIONS

### Index Packet

Provide a new Computer Generated Record format that better addresses the need for random access indexing. The Recording Index Format 3 that was previously proposed to the TG [2], shown in Figure 5, would have adequately addressed this need although a simpler format would also suffice.

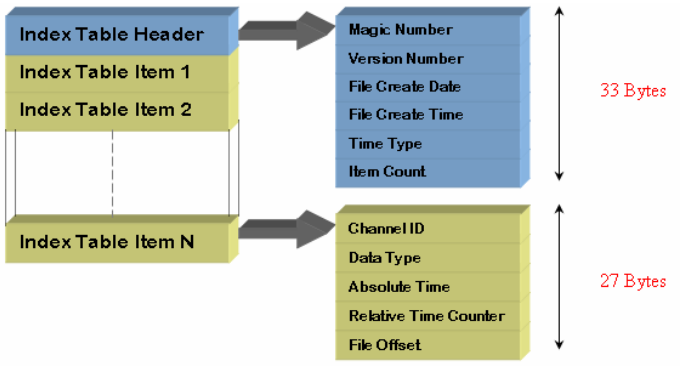


**Figure 5 Proposed Recording Index Format**

### RTC/Secondary Packet Header

Eliminate the Secondary Packet Header and provide the absolute TOD as an additional field. This will make the Packet Header easier to process and allow a simpler process for the extraction of time.

### Packet Flags

Increase the Packet Flags from an 8-bit to 32-bit field. It is inevitable that this field will have to be expanded to support future changes to the Chapter 10 standard.

### Data Type

Increase the Data Type field from 8-bits to 16-bits. This will practically ensure that the Chapter 10 standard will be able to accommodate an increase in the number of supported data types.

### Sequence Number

Increase the Sequence Number from an 8-bit to either a 16- or 32-bit field. 16 bits should be adequate to minimize sequence number repeats and 32 bits would virtually guarantee against repeated numbers.

### Checksums

Eliminate the data checksum, change the header checksum to a packet checksum, and allow CRC-16 as a checksum algorithm option. This eliminates a field from the packet trailer and can better ensure data integrity.

### Proposed Header Format

A proposed packet header format that incorporates the previous recommendations (minus the index packet recommendation) is shown in Figure 6. This new format increases the header size from 24 to 40 bytes, assuming a 16-bit sequence number. One of the bits in the Packet Flags field would be used to specify the Packet Checksum algorithm (additive or CRC).

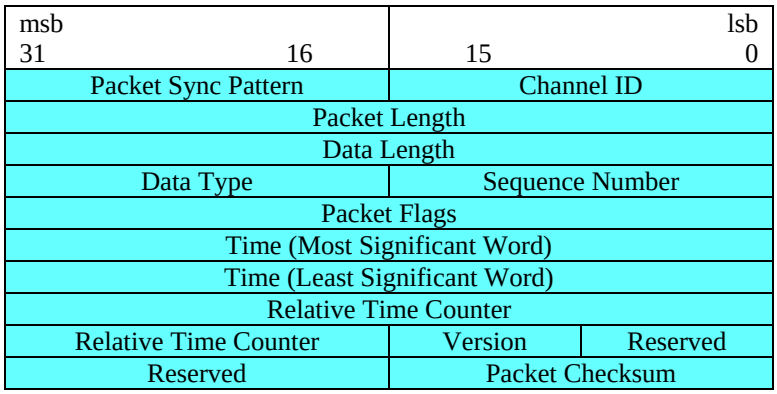


**Figure 6 Proposed packet header format**

## CONCLUSION

The IRIG 106 Chapter 10 standard is a big leap forward for the telemetry community. Chapter 10 is an open, flexible and powerful architecture to support the transition from tape-based to solid state recording devices. This standard does, however, have its shortcomings—no data standard is ever perfect—but these shortcomings can be overcome as Chapter 10 evolves to better meet the needs of the telemetry community in the 21st century.

## ACKNOWLEDGEMENTS

The author would like to thank Bill James for keeping everyone informed on the progress of the Chapter 10 standard, Min Kim and Phyllis Lake for their work on the 96th CG Chapter 10 application RAPTR, and Charlie Churillo for being such a patient and helpful customer.

## REFERENCES

[1] Telemetry Group – Range Commanders Council, IRIG 106, Chapter 10 – Solid State on Board Recorder Standard, U.S. Army White Sands Missile Range, New Mexico, 2004.

[2] Telemetry Group – Recorder/Reproducer Committee, TG-109 meeting, Tyndall Air Force Base, Florida, 24 – 26 February 2004.