GOES HDR

Binary Message Protocol

Thoughts & Suggestions

12/08/2006

Microcom Design, Inc.
10948 Beaver Dam Road, Suite C
Hunt Valley, MD 21030

Table of Contents

List of Figures

List of Tables

1 Introduction

As a result of the GOES Manufacturer's Meeting in Asheville, NC on November 9, 2006, Microcom Design, Inc. has prepared this document to share our thoughts on a GOES Binary Message Protocol and Format.

Significant discussion has taken place with regard to a new binary format, and certain key items seem to have reached a general consensus (e.g. the use of a message length instead of an EOT sequence). However, the discussion so far has been focused more at the structure level than the information or format level. While this white paper will also address some areas of the binary message structure, the primary purpose of this document is to offer a few suggestions on mechanisms to compact the information to be transmitted.

More importantly, the suggested compaction algorithms are intended to provide reduced message length without requiring any modifications in information processing. Specifically, Microcom is proposing that the binary protocol include information compaction schemes that will be "IT Transparent". Note that the proposal is to include these schemes in the standard in such a way as to not preclude generic binary message data, but as a way to facilitate a rapid transition to use of binary messages. The sole purpose of which is to reduce the message length in a GOES transmission, thereby improving the time efficiency of the system.

1.1 Binary GOES Message

While it is generally agreed that a binary GOES message format is necessary to reduce message lengths/durations in order to make better use of the limited resources, maximum improvement from a binary message will only be achieved when data points are sent using as few bits as absolutely necessary. Further, to realize this potential requires breaking the byte boundaries and bit packing data points.

ARGOS transmissions are a good example of this approach. ARGOS messages are limited to a maximum message length of 256 bits (32 bytes). Consider the simple example shown in Table 1, sending the values in binary without bit packing requires 12 bytes while bit packing the values requires only 8 bytes. Since quite often multiple values for each parameter would typically be transmitted, the total duration of a typical GOES message will be significantly reduced. Assuming that four 15 minute samples are transmitted once an hour, the information portion of a packed message would be 0.85 seconds versus 1.28 seconds unpacked.

Note that for a Pseudo-Binary (PB) message using the same parameters from Table 1 is only slightly longer than the binary unpacked message. For all but the Humidity parameter, the number of binary bytes required to represent the value is the same as the number of PB characters; the 7 bits required for humidity requires 2 PB characters.

Therefore as Table 1 indicates, it's quite possible that without bit packing binary data, little, if any, message length savings can be realized versus Pseudo-Binary GOES Messages. However, packed binary messages can have a significant back-end cost to the end users since the decoding software will have to be modified to handle the bit packing of the data points. While this is not a difficult task to implement in software, still

costs are costs and the users may opt not to incur these charges, especially since many users prefer to have their messages in a readable format anyway.

Hence there are compelling reasons to define a binary message structure that provides significant reductions in message length, but does not affect the Information Technology (IT) currently processing the GOES messages, i.e. a structure that is "IT Transparent". However, this binary format definition should not eliminate or prohibit the use of true binary GOES messages.

Table 1: Bit Packing Example

Parameter	Units	Range	Resolution	Bits	Binary Bytes	PB Bytes	ASCII Bytes
Rain Fall	counts	0 to 4095	1	12	2	2	4
Air Temp	Celsius	-40 to +60	0.1	10	2	2	5
Humidity	% rel	0 to 100	1	7	1	2	3
Solar Rad	w/m^2	0 to 2000	1	11	2	2	4
Wind Dir	deg	0 to 360	1	9	2	2	3
Wind Speed	m/s	0 to 40	0.1	9	2	2	4
Battery	volts	9 to 15.3	0.1	6	1	1	4
			Total:	64 bits 8 bytes	96 bits 12 bytes	104 bits 13 bytes	216 bits 27 bytes

1.2 IT Transparency

As noted above, the primary goal of an IT Transparent binary format definition is to reduce message durations with minimal, if any, impact on the end user's information processing software. To achieve the second half of this goal, it is therefore necessary to ultimately deliver the GOES message data to the end users in the same (or reasonably similar) format to what they now get. Since the only place to achieve the first half is at the GOES transmitter, it therefore makes sense to consider a compression or compaction scheme that is applied just prior to transmitting the data while the decompression or de-compaction is applied as soon as the message is received, i.e. at the demodulator.

If the compression/compaction process occurs in this manor, it will be completely transparent to the users. The data to be transmitted can still be buffered in the same format currently used and the same format will be used to deliver the data.

1.3 Compaction versus Compression

In the preceding section, the terms compaction and compression were used to define the process whereby the message length would be reduced prior to transmission. While a complete discussion of compression techniques is well beyond the scope of this document, there are essentially two broad categories of compression techniques, lossy and loss-less compression. Lossy compression techniques are used in video and audio

(e.g. JPEG, MP3) systems where a certain amount of information loss can be tolerated in exchange for increased compression. Obviously, lossy compression is not applicable to GOES data transfers.

Loss-less compression techniques ensure that the original data can be exactly recovered from compressed data. Very generally speaking, there are two main types of loss-less compression mechanisms; run-length encoding and entropy encoding. Run-length encoding schemes typically are most applicable to data that includes long runs of identical information (e.g. voice with long periods of silence between words). Entropy encoding schemes utilize various algorithms that characterize the data and defines character sets or tables whereby the most common piece of information or symbol is represented with only a few bits and less common characters are represented with more bits; one of the earliest uses of entropy encoding is Morse code.

The most widely known/used compression tool is PKZIP developed by PKWARE. In it's original form PKZIP used a variety of methods to compress computer files. However, in practice, the latest version of PKZIP uses Phil Katz's (the PK in PKZIP) DEFLATE algorithm. While it has been suggested to define a binary standard that uses PKZIP as the compression algorithm, using PKZIP as the tool to compress short messages (or small files) will more than likely inflate the size of the message. For example, consider the real Pseudo-Binary message data captured from channel 49 shown below.

```
B1K@Gj@Gh@Gg@Gf@Gf@Gd@Gc@Ga@EX@EX@Sw@Sw@Sw@SwH
```

Figure 1: Typical Pseudo-Binary Message

When this 46 byte sequence is saved as a text file and run through PKZIP, the resultant zip file is actually 144 bytes (over 3 times as large). However, a good portion of this is overhead as the PKZIP program is intended to compress computer files and therefore store path and filename information. On the other hand, this simple example shows that while the compression algorithm implemented within PKZIP may prove useful at some time, the actual use of PKZIP as the compression/decompression tool is not appropriate to this application. Further, even examining the actual size of the compressed data within the file shows only a 27% reduction (34 versus 46 bytes). This is only a slight improvement (1 byte) over the Pseudo-Binary Compaction scheme defined in Section 2.1.

Returning the general cases of compression algorithms and the application at hand, what can be considered a special case of entropy encoding is a reduced character set algorithm. For example, rather than entropy encoding the entire ASCII character set, reduce the number of allowed characters and encode them in a fewer number of bits. Since the resultant character set will contain less than 8 bits, the message characters can be compacted (or packed) into fewer bytes than would be necessary otherwise. While an appropriate character set or sets need to be defined for the ASCII, the idea of compaction is immediately applicable to the Pseudo-Binary format.

Further, one important caveat to the decision on the determination of any compression or compaction scheme to be utilized in GOES messages must also be considered. The primary purpose of the idea of compressing GOES messages is to reduce the message window sizes by reducing message durations. However, since it is an absolute

requirement that the message fit within the reduced window, the compression or compaction algorithm utilized must give readily predictable results. Otherwise, it is not possible to reduce the window length with 100% assurance that the compressed message will fit within the reduced window. In other words, the window length must be sized for the minimal amount of compression.

To clarify this caveat, consider the use of run-length compression as applied to a series of sensor readings that have not changed for a period of time (e.g. tipping bucket counts during extended period of no rainfall). A simple count of the repeated values would allow the message length to be reduced by sending the count and only one copy of the value. While this would reduce the message length for certain transmissions, the window size could not be reduced since the window must be sized to accommodate messages when the data does change frequently (e.g. on rainy days for tipping bucket readings).

Note there are certainly other good reasons to implement compression techniques that produce shorter messages sometimes (e.g. reduced power consumption, reduced susceptibility to interfering transmitters, etc.). However, the primary goal of reduced message windows is not satisfied by these algorithms.

Microcom believes that the simple compaction techniques suggested in the subsequent sections of this document will not only be easier to implement, but will also prove to be effective compression algorithms for the special cases of GOES messages.

2 IT Transparent Compaction

Microcom is proposing three compaction schemes. In general, these schemes are intended to produce more efficient use of message windows without burdening the users with costly software upgrades/modifications to their information processing systems; i.e. the schemes are IT Transparent.

The schemes suggested are geared toward users who are already reducing messages durations using Pseudo-Binary coding, as well as user's who are reluctant to switch from transmitting and receiving ASCII data.

2.1 Pseudo-Binary Compaction

Pseudo-Binary is actually a good first step toward reducing GOES message lengths and is, by its nature, a compaction scheme already being employed by many GOES users.

Microcom's Pseudo-Binary Compaction scheme is a simple extension of this format that would reduce the information portion of the message by 25%.

In Pseudo-Binary (PB) format, binary data is encapsulated in ASCII characters. Each ASCII character can carry up to six bits of binary information. To express values beyond 6-bits, multiple ASCII characters are simply concatenated. In other words, two PB characters can represent 12-bits of information, 3 characters can represent 18-bits, and so on.

To ensure the ASCII characters used to convey the binary information are printable, the PB character definition requires bit 6 of the byte to be a 1 (one); bit 7 is used for the odd parity byte as shown below.

$$P_O \; 1 \; B_5 \; B_4 \; B_3 \; B_2 \; B_1 \; B_0$$

Figure 2: Pseudo-Binary Bit Map

With this definition, the PB character set extends from 0x40 ('@') to 0x7F (DEL). Since DEL is the only non-printable character in this range, the PB standard allows 0x3F ('?') to be sent instead (the only time B_6 can not be 1). However, the important point is that only B_5 through B_0 actually carry the information of concern to the end-user.

As such, a simple mechanism to reduce the message length is to transmit just the information bits. Further, this compaction could be done on the fly by the transmitter, and the de-compaction could be performed by the demodulator as it is being received. Since this process is limited to the transmitter/receiver pair, the result is essentially a 25% reduction in message information length that is transparent to the user.

In the example shown below, the first four characters of the Pseudo-Binary message from Figure 1 are compacted into three bytes. Note that as shown in Figure 3 the same 24 information bits (4 x 6 = 24) are sent (the parity and forced 1 bit are not information bits) from the original four bytes, just concatenated into three bytes (3 x 8 = 24)

```
P1--B---  P1--1---  P1--K---  P1--@---
11000010  00110001  11001011  11000000

      --B--- --1--- --K--- --@---
      000010 110001 001011 000000
      xxxxxx xxyyyy yyyyzz zzzzzz

       00001011 00010010 11000000
       xxxxxxxx yyyyyyyy zzzzzzzz
```

Figure 3: Pseudo-Binary Compaction

Applying this compaction scheme to the entire message of Figure 1, would reduce the information from 46 bytes to 35 bytes. For a GOES HDR 300 bps message this is a message length reduction of almost 0.3 seconds.

At the receive end the demodulator would know this compaction algorithm was in effect, and simply reverse the procedure and deliver the end user the original pseudo-binary data shown in Figure 1.

2.2 Numeric ASCII Compaction

While the Pseudo-Binary compaction scheme can yield a 25% reduction in information message duration, the Numeric ASCII Compaction scheme can yield a 50% reduction. Microcom has noted that many users are still sending ASCII, but are only sending the data values, i.e. no labels or other text strings are being sent. For these user's only a small portion of the entire ASCII character set is being transmitted, primarily the ASCII numerals 0 through 9. Two examples of such messages captured from a GOES DRGS as shown below.

```
08.38
009
135
-02
000
085
15.9
020.7
144
014
00161
```

Figure 4: Numeric ASCII Example 1

Using these two examples as a basis, Microcom has defined a 4-bit character set that could be used to encode these messages as shown in Table 2. Using this character set and compacting two characters into every byte, will yield a 50% reduction in the information portion of the message.

```
+2006.,+337.0,+1856.,+13.31,+13.57,+14.15,+4940.,+29.62,+720.0,
+13.50,+10.00,+4.000,+2.400,+13.10,+10.00,+0.000,+27.07,+412.1,
+19.39,+15.31,+79.14,+25.08,+721.5,+21.94,+21.94,+2.205,+31.20,
+46.64,+1089.,+13.37,+24.63,+120.0,+0.207,+4.578,+10.13,+458.3,
+13.46,+23.13,+120.0,+0.143,+4.513,+13.36,+333.3,+13.44,+24.59,
+120.0,+56.33,+28.25,+0.527,+1542.,+34.89,+30.14,+10.00,+56.63,
+28.26,+5.468,+1542.,+35.09,+5.601,+10.00,+56.45,+28.25,+1542.,
+34.96,+10.00,+20.60,+14.90,+59.62,+25.16,+67.09,+412.0,+0.000,
+10.00,+0.000,+0.000,+0.000,+0.000,+0.000,+0.000,+13.58,+120.0,
```

Figure 5: Numeric ASCII Example 2

Note that the numeric character set shown in Table 2 does not include a cr/lf sequence. To accommodate this sequence as well as a few other possible (albeit less used) numeric characters, the special code sequences in Table 3 are also defined. Since these combinations of polarity symbols and the decimal point are not valid in numeric representations, they can readily be implemented to effectively increase the character set with minimal, if any, loss of efficiency. Specifically, using only the cr/lf sequence does not affect the overall compression as two ASCII characters are still compressed to two 4-bit codes.

Table 2: Possible Numeric ASCII Character Set

Char	Code	Char	Code	Char	Code	Char	Code
0	0000	4	0100	8	1000	. (dp)	1100
1	0001	5	0101	9	1001	+	1101
2	0010	6	0110	space	1010	-	1110
3	0011	7	0111	,	1011	/	1111

Table 3: Numeric ASCII Special Character Translation

ASCII Char(s)	Numeric Chars	Double Code
cr/lf	++	1101 1101
#	+-	1101 1110
=	-+	1110 1101
:	..	1100 1100
E	--	1110 1110

The improvement and simplicity of this compaction scheme are readily apparent. While additional reductions in message lengths can be realized for these types of messages, this is more a function of the user community than any compaction or compression scheme. For example, in Figure 4, the cr/lf sequences terminating each value could

simply be replaced with spaces or commas. By simply replacing the two-character termination (cr/lf) with a single character termination (e.g. space), an additional 5 bytes can be eliminated.

In the second example, every value is terminated with a comma. A simple count will show that there are 81 commas embedded in this message. While comma delimited values are readily imported into spreadsheets, this process could readily be done by a simple software tool that scans the message after receipt and inserts a comma ahead of every polarity sign. By adopting a convention similar to the familiar SDI-12 protocol, the plus and minus symbols could suffice as the numeric separators.

On the other hand, note that all the values in the example of Figure 5 are positive. If the vast majority of these readings can not be negative, simply omitting the plus sign for these values can produce a similar result without impacting how the comma delimited string shown is imported into a spreadsheet. Note also that some values are terminated in a decimal point; if these values are always whole numbers, than transmitting this character is unnecessary.

In summary, by tweaking the original message (users) and implementing the ASCII Numeric Compaction algorithm (manufacturers), significant message length reductions can be readily achieved without significant effort.

2.3 Alphanumeric ASCII Compaction

The Numeric Compaction scheme, described in the preceding section can be readily extended to a more complete Alphanumeric ASCII Compaction scheme. Microcom is suggesting an alphanumeric compaction algorithm that is based on limiting the alpha portion of the character set to capital letters only.

The approach makes use of entropy encoding by using 5 bits to represent the most common characters (e.g. the numeric symbols of Table 2), and defining a 6-bit character set for the letters and some additional punctuation characters.

The Alphanumeric ASCII Compacted character set is shown in Table 4. The first column of Table 4 is the numeric characters from the Numeric ASCII character set. However, these characters are now encoded as a 5-bit binary value with the most significant bit being 0. An additional 31 characters (including the uppercase letters) is defined using a 6-bit code that has the most significant bit set to 1. This variable size code set provides a total of 47 characters.

During the compaction process, the 5 or 6-bit codes are continuously packed together to form bytes. The compacted data is then transmitted. As the data is received, the codes are de-compacted and reverse translated. The de-compaction algorithm first requires the examination of the next un-compacted bit to determine how many total bits to extract (0 = 5 bits and 1 =6 bits).

Assuming all characters are equally likely to be represented, the average number of bits per character is 5.66. Therefore, the minimum compression ratio is 29.3%. However, assuming the data to be compacted is totally numeric, a compression ratio of 37.5% is

possible; although, if this were the case Numeric ASCII compression should be used. Regardless, the effective compression ratio is between 29% and 37% depending on the data to be compacted. Since the message format is known by the user, a simple calculation can determine the compression for a given message structure. For example, assuming 10% of the bytes are alpha and 90% of the bytes are numeric, the average compression is 36.3% (i.e. (8-5.1)/8).

Table 4: Alphanumeric ASCII Character Set

Char	Code	Char	Code	Char	Code
0	00000	A	100000	Q	110000
1	00001	B	100001	R	110001
2	00010	C	100010	S	110010
3	00011	D	100011	T	110011
4	00100	E	100100	U	110100
5	00101	F	100101	V	110101
6	00110	G	100110	W	110110
7	00111	H	100111	X	110111
8	01000	I	101000	Y	111000
9	01001	J	101001	Z	111001
space	01010	K	101010	cr/lf	111010
,	01011	L	101011	#	111011
.	01100	M	101100	=	111100
+	01101	N	101101	:	111101
-	01110	O	101110	;	111110
/	01111	P	101111	n/a	111111

3 Binary Protocol Comments

While the previous section focused on suggested binary compaction formats that are somewhat independent of the binary protocol, there are several topics with regard to the message structure that should be given consideration in developing a complete binary protocol. Further, Microcom has given some additional thought to the binary protocol subsequent to the discussions at the recent GOES Manufacturers Meeting in Asheville.

3.1 Length versus EOT

Microcom agrees with what appeared to be the consensus of opinion at the Asheville meeting that a message length should be used in place of an EOT sequence. Use of any type of EOT sequence is problematic (bit stuffing or byte stuffing) to prevent the sequence from appearing in the message portion. Further, as was noted in Asheville, the length field provides other advantages and is no more costly in transmission time then the EOT sequence.

3.2 Character Length versus Byte Length

While an overall byte length for the message can be included, the compaction schemes presented in this paper will require a character length in addition to, or in place of, an actual byte length. The reason for this is that the compaction schemes have multiple characters (or portion thereof) residing in a single byte and/or characters crossing byte boundaries. Accordingly, to distinguish between real information content and unfilled data, a character length is required.

The simplest example of this is the Numeric ASCII compaction. At the end of the reception process when the last byte of data has all zeroes in the four least significant bits, the receiver needs some way to know whether this is an unused character or is the digit ‘0’.

While not as readily apparent, a similar situation also arises in the Pseudo-Binary case when three pseudo-binary characters must be compacted, i.e. when the overall character length is one byte less than an integer multiple of 4. The three PB characters will still require 3 bytes for compaction, but the receiver needs to know that the final six bits should not be de-compacted; otherwise, an extra ‘@’ symbol will be generated (assuming a zero fill – the ‘@’ symbol is the PB character for the all zero case).

Therefore, some mechanism must be included to identify the character length. Note that even though the converse is not true, knowing the character length does guarantee the determination of the exact number of bytes required for a given compaction scheme.

3.3 Packet Length and CRC Check

It was also discussed in Asheville that the binary protocol should include a CRC; specifically, the 16-bit CRC-CCITT. Microcom has used similar 16-bit CRCs and also agrees that such a check should be included in the binary specification.

However, Microcom believes that a single CRC at the end of the message is not sufficient when used for longer message lengths. Microcom suggests that a Packet Length/CRC protocol be included in the specification. The Packet Length can be a

single byte value, allowing up the 256 bytes/characters per packet. At the end of each packet a 16-bit CRC should be included to verify the validity of each individual packet.

Note that the Packet Length field can also solve the problem of the character length issue in the previous section. However, when the message is true binary, the length is in bytes.

For the numeric ASCII compaction scheme a character length of 256 requires 128 bytes (2 characters per byte). Even an addition of a 16-bit CRC is still much less parity overhead than in the current ASCII format (1 bit per 8 bytes versus 1 bit per byte), and a CRC can detect multiple bit errors in a byte more effectively than a single parity bit.

3.4 Length Field and BER versus Phase Reference Loss

During the aforementioned Asheville meeting, the topic of protecting the length from bit errors was also discussed. While Microcom originally agreed that the length should be encoded in such a way as to ensure its integrity, subsequent analysis has altered Microcom's position. At the minimum signal-to-noise (SNR) ratios for the GOES HDR system (11-12 Eb/No), the BER performance has a theoretical performance significantly better than 10^{-8}. Even factoring in the implementation loss of the *DigiTrak* demodulators currently in use at Wallops CDA, the BER has been shown to be better than 10^{-6} at the minimum SNR.

Further, as was noted in Asheville, the primary limiting factor in GOES message reception at the minimum SNR is not BER, but instead Phase Reference Loss. Upon receipt of a GOES message, the carrier portion of the message is used to establish the initial phase reference of the signal. However, the reference phase of the GOES signal does not remain constant during the message, and the demodulators must attempt to maintain phase reference in the presence of modulation, noise, and frequency drift. At high SNRs, this is a fairly easy and reliable process. However, at the SNR drops toward the minimum level, the ability to maintain phase reference approaches a threshold.

Above this threshold, the demodulators have an extremely high probability of maintaining phase reference. However, Microcom has shown that below the threshold the probability of phase reference loss increases significantly and is directly related to message length. This threshold phenomenon occurs at approximately 12.5 dB Eb/No for 300 bps messages and at 11.5 dB Eb/No for 1200 bps. DAMS-NT acceptance testing showed that at these levels the demodulators should have a BER performance better than 10^{-7}.

Once phase reference is lost, BER is no longer meaningful as the phase information is being correlated against the wrong phase. In other words, every symbol is perceived to be erroneous. Accordingly, once phase reference is lost, the demodulated data is completely garbled, and no amount of subsequent error correction can rectify the problem.

On the other hand, this also implies that the least likely place to have garbled data is at the beginning of the message. Therefore, as long as the overall byte length is transmitted early in the process, it should have a high reliability of being properly received.

Potential loss of phase reference is also a good argument for utilizing smaller packets with a CRC per packet as was suggested in the previous section. If only a single CRC is used on long messages (> 1000 bytes), then loss of phase reference at the tail end of the data will make it impossible to validate any of the data. Breaking the message into packets with a CRC per packet would provide an opportunity to validate a portion of the data even if phase reference is lost downstream.

It has also been suggested that the message length be encoded in the same 31/10 BCH code used for the GOES ID. However, based on the preceding discussion, this does not appear to be necessary. Further, the BCH code used for the GOES ID, can theoretically only correct 2 bit errors. However, the correction algorithm provided to Microcom that was implemented in the DAMS-NT system can not correct every combination of two-bit errors. Further, a side nature of the Trellis encoding used in the GOES system is that when bit errors do occur they tend to come in bursts; this is a result of the fact that when the Trellis decoder can not correct an error due to excessive noise, it identifies an erroneous correction path that typically will alter several bits. In other words, if an error occurs in the BCH code, it's more likely than not to have more then 1 bit in error and quite probably more than 2.

3.5 Phase Reference Retraining

As was noted in the previous section, at low signal-to-noise ratios, the primary cause of corrupted data is loss of phase reference at the demodulator. This naturally begs the question: Is it possible to improve the phase tracking performance?

In theory, the answer is "yes". A common technique employed in digital cell communication is to periodically send a known sequence of modulation to allow the receiver to retrain or re-establish a good phase reference.

During the DAMS-NT project, test results showed that at 11 dB Eb/No, a 10-second, 300 bps HDR message has a probability of not losing phase reference and being fully received of ~95%. However, a 2.5 second message at the same SNR is properly received virtually 100% of the time. Further, as the message length increases the probability of completely receiving the message falls exponentially. This suggests that if a solid phase reference can be re-established every couple of seconds, the probability of correctly receiving long messages can be significantly improved. This is because a single long message would essentially become a series of concatenated short messages.

The unknown factor in this approach is how much retraining time is required. Obviously, one can surmise that by inserting 0.5 seconds of carrier followed by 18 symbols of bi-phase data (3 clocks and a 15-bit FSS) every 2.5 seconds, a 10-second message should have similar performance to a 2.5 second messages. However, adding 0.6 seconds of transmission time every 2.5 seconds is obviously contrary to the desire to reduce message lengths, and is too costly from this perspective. On the other hand, if the retraining time could be reduced to a tenth of a second or less and still produce similar results, perhaps it would be a worthwhile cost.

The next logical question is how to effectively insert a retraining sequence. Obviously, a simple approach would be to insert the sequence at a fixed time interval (e.g. after

every 2.5 seconds of modulated data). However, the final retraining sequence may prove wasteful if there are only a few bytes left to transmit; for example, adding a retraining sequence for a 2.6 second message is most likely of very little use. Further, setting a fixed time period is somewhat limiting since it's not adaptable to the variety of user messages used in GOES DCS.

In the next section, a proposed binary message structure based upon packets is detailed. Using a packet structure greatly facilitates the possibility of inserting a retraining sequence as the sequence can be readily inserted after all but the last packet to be transmitted.

Unfortunately, a clear recommendation cannot be made as to whether or not this process would prove useful without being too costly (from a message duration perspective). Additional analysis and experimentation would be required to fully characterize the benefits and costs of including a retraining sequence. However, it certainly seems as though this could prove useful and would be a worthwhile investigation.

4 Binary Protocol Summary

This section summarizes some of the more salient points for a suggested binary message protocol to be implemented in the GOES system. While this is only a summary, complete details of proposed GOES HDR Binary Protocol Specification is provided in a separate document. In addition to detailing the proposed message structure, the specification also encapsulates the compaction schemes explained previously.

4.1 General GOES HDR Binary Structure

The proposed general binary message structure is shown in Figure 6 and Figure 7. Note that there are actually two formats proposed; one for short messages and one for longer messages. The primary distinction is that the longer messages will be transmitted in two or more packets, and short message will only require a single packet.

Carrier 0.5s 0.25s	Clock 1-0-1 1=180	FSS 15 Bits	GOES ID 4 Bytes	Flag Byte 8 bits	Packet Length 1 Byte	Data	16-Bit CRC	Encoder Flush (Variable)

Figure 6: Binary Message Structure Single Packet

Carrier 0.5s 0.25s	Clock 1-0-1 1=180	FSS 15 Bits	GOES ID 4 Bytes	Flag Byte 8 bits	Message Length 2 Bytes	Data Packets | Length | Data | CRC |	Encoder Flush 2 Bytes

Figure 7: Binary Message Structure Multiple Packets

The basic packet structure is shown in Figure 8. Each packet consists of a one byte length field, a variable number of data bytes (as defined by the length field), and a 16-bit CRC. Allowing for variable length packets allows the packet sizes to be tailored or optimized based on an individual user's requirements.

Packet Length 1 Byte	Data Bytes	16-Bit CRC

Figure 8: Basic Packet Structure

As shown in Figure 7, longer messages will require multiple packets to completely send the data. When the multiple packet structure is used, an overall message length field is included prior to the first packet. The proposed message length field has the format shown in Figure 9.

$P_O L_{13} L_{12} L_{11} L_{10} L_9 L_8 L_7$ MS Byte	$P_O L_6 L_5 L_4 L_3 L_2 L_1 L_0$ LS Byte

Figure 9: Message Length Structure

Additional details on the proposed binary message structure and how it is implemented with the compaction algorithms are provided in the aforementioned proposed GOES HDR Binary Protocol Specification.

4.2 GOES HDR Flag Byte

One natural question that arises from the compaction algorithms discussed in Section 2 is how to identify the actual compaction algorithm utilized in a message. Microcom suggests the following use of the flag bits in the GOES HDR flag byte to address this requirement.

Table 5: GOES HDR Flag Byte (Modified)

Bit(s)	Name	Description
1 LSB	Multiple Packets B_{MP}	0 = Message includes only one packet. 1 = Message includes multiple one packet. Only used for Binary and Compacted Messages.
2	UTC Time Sync B_{TS}	0 = No UTC Time Sync since last transmission. 1 = UTC Time Sync since last transmission. Only used for Self-Timed Messages.
3	Compaction B_{CM}	0 = No Compaction in Use. 1 = ASCII/Pseudo Binary Compaction Active. (Binary application to be determined.)
4	Coding Type B_{CM}	0 = ASCII Numeric Compaction. 1 = ASCII Alphanumeric Compaction. (Binary and Pseudo-Binary use to be determined.)
5	Spare	TBD – Send as 0
7/6	Message Type B_{MT}	00 = Reserved. 01 = ASCII 10 = Binary 11 = Pseudo-Binary
8 MSB	Parity P_O	Odd Parity

Additional details on the meaning and use of these flag bits are provided in the aforementioned proposed GOES HDR Binary Protocol Specification.