



## 1. Introduction

This specification defines the proposed HRIT file format to replace the existing LRIT/HRIT DCS File format.

## 2. HRIT DCS File Naming

The filename for HRIT DCS messages will as follows:

pH-YYDDDDHHMMSS-Q.dcs

where,

- H designates the new HRIT file format
- YYDDDDHHMMSS is the file date/time of creation in UTC Julian format
- Q is an ASCII letter (A to Z) used in the event two files are generated at the same time

## 3. HRIT DCS File Structure

The BNF HRIT DCS file structure is summarized below:

FILE := FILE\_HEADER BLOCKS FILE\_CRC32

FILE\_HEADER := FILE\_NAME FILE\_SIZE SOURCE TYPE EXP\_FIELD HDR\_CRC32

FILE\_NAME := 32\*octet

FILE\_SIZE := 8\*octets

SOURCE := 4\*octets

TYPE := 4\*octets (:= "DCSH", see Section 3.1.4)

EXP\_FLD := 12\*octets

HDR\_CRC32 := 4\*octets

BLOCKS := BLK\_ID BLK\_LNG BLK\_DATA BLK\_CRC

BLK\_ID := 1\*octet

BLK\_LNG := 2\*octets

BLK\_DATA := 0..65,530\*(octets)

BLK\_CRC16 := 2\*octets

FILE\_CRC32 := 4\*octets

DCP\_BLOCK := DCP\_ID BLK\_LNG DCP\_HDR DCP\_DATA BLK\_CRC (see Section 3.3)

DCP\_ID := 1\*octet (:= 0x01)

BLK\_LNG := 2\*octets

DCP\_HDR := 36\*octets (see Section 3.3.1)

DCP\_DATA := 0..16,000\*(octets)

BLK\_CRC16 := 2\*octets

MM\_BLOCK := MM\_ID BLK\_LNG MM\_HDR BLK\_CRC (see Section 3.4)



MM\_ID := 1\*octet (:= 0x02)  
 BLK\_LNG := 2\*octets  
 MM\_HDR := 24\*octets (see Section 3.4.1)  
 BLK\_CRC16 := 2\*octets

### 3.1. File Header

The FILE\_HEADER precedes all DCS message and only occurs once. The File Header consists of several fields as summarized below and detailed in the following subsections.

FILE_NAME	Original HRIT filename
FILE_SIZE	File Length in octets/bytes
SOURCE	Site source code for the file (i.e. WCDA or NSOF)
TYPE	Internal DCS file type (not the same as the LRIT/HRIT file code)
EXP_FIELD	Expansion Field 2 – Reserved for Future Use
HDR_CRC32	32-bit Header CRC check

#### 3.1.1. FILE\_NAME := 32\*octet

The FILE\_NAME field is a 32-byte field containing the DCS message filename as defined in Section 3.6. The filename is left-justified, and the remainder of the field will be filled with space characters.

#### 3.1.2. FILE\_SIZE := 8\*octets

An 8-character field with the ASCII representation of the total file size in bytes. Left-justified and space filled.

#### 3.1.3. SOURCE := 4\*octets

A 4-character field identifier specifying the source of the HRIT DCS data.

Data sourced from Wallops Command and Data Acquisition station will use the code “WCDA”.

Data sourced from NOAA’s Satellite Operations Facility will use the code “NSOF”.

Future site source designations will be defined as necessary.

#### 3.1.4. TYPE := 4\*octets (:= “DCSH”)

A 4-character field identifier defining a DCS file type. This field will use the code word “DCSH”.

Note that this type designation is not the same as, and is in addition to, the LRIT/HRIT header file type, which is a byte value of 0x82.

#### 3.1.5. EXP\_FILL := 12\*octets

A 12-character text field reserved for future use. This field is space filled.

#### 3.1.6. HDR\_CRC32 := 4\*octets

The HDR\_CRC32 is a 32-bit CRC generated per IEEE RFC 1952 on the first 60 bytes of the FILE\_HEADER, i.e. the FILE\_NAME thru EXP\_FIELD sections. The value is stored in the file as binary values in little-endian format.

See Section 3.6 for code samples on computing the CRC.



## 3.2. BLOCKS

Following the file header, the remainder of the file will include one or more data concatenated BLOCKS; the FILE\_CRC32 field (see Section 3.4) is appended after the last data block.

All data BLOCKS will have the following general format:

BLK_ID	Block data type identifier
BLK_LNG	Unsigned integer value of block length (total bytes all fields)
BLK_DATA	Block data (0 to 65,530 bytes)
BLK_CRC16	16-bit CRC on all preceding bytes in the block.

To support future enhancements while maintaining backward compatibility, this proposed HRIT file format utilizes a block identifier (BLK\_ID) immediately followed by a block length (BLK\_LNG) field. When implementing the file processing code, systems should be designed to look at the block identifier to determine the type of data in the block and the appropriate handling. If the BLK\_ID is not a value the code recognizes or supports, the code should then use the length field to skip over this data.

This approach allows future data block types to be defined without negatively impacting deployed systems until the code can be updated to support the new block type.

### 3.2.1. BLK\_ID := 1\*octet

The BLK\_ID is a single byte field identifying the type of data in the block. Presently the only block type defined is the DCS message block (see Section 3.3).

Future block types may be defined as needed or desired

### 3.2.2. BLK\_LNG := 2\*octets

The block length (BLK\_LNG) field is a 2-octet unsigned integer value designating the total length of the block in octets (aka bytes). Since the value includes the identifier, the length itself, the variable data, and the 16-bit block CRC; this field can have a value between 5 and 65,535.

Note that a block length of 5 indicates no data is present in the block. Further, the maximum size of the data field is 65,530 octets.

The two byte block length value is provided in little-endian order.

### 3.2.3. BLK\_DATA := 0..65,530\*(octets)

This is the variable length data field. The actual data in the block depends on the block type.

### 3.2.4. BLK\_CRC16 := 2\*(octets)

The BLK\_CRC16 is a 16-bit CRC that can be used to validate the block. This 16-bit CRC is identical to the LRIT/HRIT packet CRC defined by the generator polynomial:

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

The CRC is initialized to “all ones” prior to the CRC calculation. All bytes beginning with the block identifier (BLK\_ID) up to and including the last byte in the data field is included in the CRC generation. The value is stored in the file as binary values in little-endian format.

See Section 3.6 for code samples on computing the CRC.

### 3.2.5. Sequence Numbers

All BLOCKS include a header, and the first field in all headers is a 3-byte sequence number that can be utilized to identify missed BLOCKS. The sequence number generation is common to all block types, i.e. there is not a separate sequence number for each block type. Even though the sequence number is part of a Block’s header, since it will



always be located immediately following the Block Length, even unknown Block types can be checked for a valid sequence number.

### 3.3. DCP ASCII and Pseudo Binary Message Blocks

DCP Message blocks are the primary data blocks in the HRIT DCS file. This section defines the format for ASCII and Pseudo Binary messages, which also include decompacted ASCII and Pseudo Binary sent in a compacted binary format

DCP ASCII and Pseudo Binary (PB) Message blocks have a block type identifier of 0x01 and have a variable length based on the DCP message header and DCP data. The DCP Message block consists of the following fields

DCP_ID	DCP Message Block ID (:= 0x01)
BLK_LNG	DCP Message Block Length (see Section 3.2.2)
DCP_APB_HDR	DCP ASCII and PB Message Header (see Section 3.3.1)
DCP_DATA	DCP Message Data (data as received from DCP or informational message)
BLK_CRC16	16-bit CRC (see Section 3.2.4)

#### 3.3.1. DCP\_APB\_HDR

The DCP ASCII and PB Message Header is a 36-byte field defined by Table 1.

**Table 1: DCP Message Header**

Field Name	Bytes	Format	Section
Sequence Number	3	Integer Unsigned	3.2.5
Message Flags/Baud	1	Bit Mapped	3.3.1.1
Message ARM Flags	1	Bit Mapped	3.3.1.2
Corrected Address	4	Hexadecimal	3.3.1.3
Carrier Start	7	BCD	3.3.1.4
Message End	7	BCD	3.3.1.5
Signal Strength X10	2	Integer Unsigned	3.3.1.6
Frequency Offset X10	2	Integer Signed	3.3.1.7
Phase Noise X100	2	Integer Unsigned	3.3.1.8
Good Phase X2	1	Integer Unsigned	3.3.1.9
Channel/Spacecraft	2	Integer Unsigned/Bit Mapped	3.3.1.10
Source Code	2	ASCII Characters	3.3.1.11
Source Secondary	2	TBD	3.3.1.12

Total: 36

### 3.3.1.1. ASCII and PB Message Flags/Baud

The ASCII and PB Message Flags field is a bit-mapped byte defined utilized as follows:

**Table 2: DCP ASCII and PB Message Flags**

B7	B6	B5	B4	B3	B2	B1	B0
					Data Rate: 001=100; 010=300; 011=1200		
					100=400; 101=800		
					000=Undefined; 110-111 Reserved		
					0=>CS1 Platform; 1=>CS2 Platform		
					1=> Message Received with Parity Errors		
					1=> No EOT Received with Message		
					Binary BCH Flag/Length Field: 1=> Corrected; 0=> Valid or not Decompacked		
					1=> Decompacked ASCII or Pseudo Binary Message; 0=> Standard Message		

### 3.3.1.2. Message ARM Flags

The Abnormal Received Message flag byte is defined in Table 3.

**Table 3: DCP Message ARM Flags**

B7	B6	B5	B4	B3	B2	B1	B0
					Address Corrected		
					Bad Address – Not Correctable		
					Invalid Address – Not in PDT		
					PDT Incomplete		
					Timing Error – Outside Window		
					Unexpected Message		
					Wrong Channel		
					Reserved for Future		

Unused or reserved bits will be set to 0.

### 3.3.1.3. Corrected Address

This Corrected Address is a 4-byte hexadecimal (binary) field providing the BCH correction of the received Platform Address. If the address is received without errors or is uncorrectable, this field will match the actual received address.

### 3.3.1.4. Carrier Start

The Carrier Start is a 7-byte BCD numeric field providing the carrier start timestamp of the message. The BCD field format is: YYDDHHMMSSZZZ where

YY	=	Last two digits of the year
DDD	=	Julian day of the year
HH	=	Hour
MM	=	Minute
SS	=	Second
ZZZ	=	Sub-second to millisecond resolution

The byte order is from least significant digit of the sub-seconds (ZZZ) to the most significant digit of the year (YY); i.e. in little endian format.

The Carrier Start is the time when the signal energy was first detected.

### 3.3.1.5. Message End

The Message End field is a 7-byte BCD numeric field providing the message end timestamp. The BCD field format is the same as the Carrier Start field defined in the previous section.

The Message End is the time when the signal energy was no longer detectable.

### 3.3.1.6. Signal Strength X10

The Signal Strength field is a 2-byte unsigned integer indicating the received message signal level in dBm EIRP. The field value is the signal level multiplied by 10; i.e. 0.1 dB resolution. The two byte value is provided little-endian.

The range of this value requires 10-bits. When processing the field, the upper six bits should be masked off to allow future use of these bits for other purposes; i.e. the data field should be masked with 0x03FF before processing.

Figure 1 shows the Signal Strength binary format. After masking off the unused bits, the resulting integer value from the least-significant 10-bits must be divided by 10.

B <sub>15</sub>	B <sub>14</sub>	B <sub>13</sub>	B <sub>12</sub>	B <sub>11</sub>	B <sub>10</sub>	B <sub>9</sub>	B <sub>8</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
Reserved for Future						Signal Strength X10									
0	0	0	0	0	0	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

**Figure 1: Signal Strength Format**

### 3.3.1.7. Frequency Offset X10

The Frequency Offset field is a 2-byte signed integer indicating the frequency offset from the channel center of the received message. The field value is the frequency offset multiplied by 10; i.e. 0.1 Hz resolution. The two byte value is provided little-endian.

The range of this value requires 14-bits including the two's complement sign bit. When processing the field, the upper two bits should be masked off and the sign bit extended to allow future use of these bits for other purposes; i.e. the data field should be masked with 0x3FFF and sign extending before processing.

Figure 2 shows the Frequency Offset binary format. After masking off the unused bits and sign extending as necessary, the resulting integer value from the least-significant 14-bits must be divided by 10.

B <sub>15</sub>	B <sub>14</sub>	B <sub>13</sub>	B <sub>12</sub>	B <sub>11</sub>	B <sub>10</sub>	B <sub>9</sub>	B <sub>8</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
Reserved		Frequency Offset X10 (2's complement)													
0	0	S	2 <sup>12</sup>	2 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

**Figure 2: Frequency Offset Format**

### 3.3.1.8. Phase Noise X100 and Modulation Index

The Phase Noise and Modulation Index field is a 2-byte entry that provides two pieces of information.

The Phase Noise field is an unsigned integer indicating the phase noise in degrees RMS of the received message. The field value is the phase noise multiplied by 100; i.e. 0.01 degree RMS resolution. The two byte value is provided little-endian.

The range of this value requires 12-bits. When processing the field, the upper four bits should be masked off to get just the Phase Noise value; i.e. the data field should be masked with 0x0FFF before processing.

The two most significant bits provide the Phase Modulation Index in the legacy three-level approach; Normal, High, or Low. The remaining two bits (B<sub>13</sub> and B<sub>12</sub>) are reserved for possible future use.

Figure 3 shows the Phase Noise Phase Modulation Index binary format. For the Phase Noise value, after masking off the four most significant bits, the resulting integer value from the least-significant 12-bits must be divided by 100.



B <sub>15</sub>	B <sub>14</sub>	B <sub>13</sub>	B <sub>12</sub>	B <sub>11</sub>	B <sub>10</sub>	B <sub>9</sub>	B <sub>8</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
Mod Index	Reserved			Phase Noise X100											
Table 4	0	0		2 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

**Figure 3: Phase Noise and Modulation Index Format**

The two most-significant bits provide the Modulation Index code as defined in Table 4.

**Table 4: Modulation Index**

Code	Meaning
00	Unknown
01	Normal or 'N'
10	High or 'H'
11	Low or 'L'

3.3.1.9. Good Phase X2

The Good Phase is percentage score indicating the quality of the received message. This is a single byte unsigned integer value with a resolution of 0.5%. Messages received with a Good Phase score of 85% or higher are considered "good". A good phase score between 70% and 85% is considered fair, and below 70% is considered poor.

The percentage scores of 70% and 85% roughly correlate to BER estimates of 10<sup>-4</sup> and 10<sup>-6</sup>. In other words, a Good Phase score of 85% or higher indicates the BER is 10<sup>-6</sup> or better, while a score of 75% or lower indicates a BER of 10<sup>-4</sup> or worse.

The range of this value requires all 8-bits of the byte field. The 8-bit unsigned integer value must be divided by 2.

3.3.1.10. Channel/Spacecraft

The Channel and field provides the DCS channel the messaged was received on along with an indication of which GOES satellite the message was received from as provided by the ground station.

While generally speaking, the satellite should be the GOES spacecraft the platform is assigned to, there are rare instances this will not be the case. For example, there have been rare occasions where one of the GOES satellites has failed, and the corresponding receive system has utilized the other spacecraft to continue operations.

Figure 4 shows the Channel/Spacecraft binary format. The least-significant 10-bits provide the unsigned integer value for the received channel. Presently the DCS channels range from 1-266 and 301-566.

B <sub>15</sub>	B <sub>14</sub>	B <sub>13</sub>	B <sub>12</sub>	B <sub>11</sub>	B <sub>10</sub>	B <sub>9</sub>	B <sub>8</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
Spacecraft				Reserved		Channel Number									
See Table 5				0	0	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

**Figure 4: Channel/Spacecraft Format**

The four most-significant bits provide the spacecraft or satellite (e.g. E, W, etc.) code as defined in Table 5.

**Table 5: Spacecraft Codes**

Code	Spacecraft
0000	Unknown
0001	GOES-East or 'E'
0010	GOES-West or 'W'
0011	GOES-Central or 'C'



Code	Spacecraft
0100	GOES-Test or 'T'
0101-1111: Reserved for Future	

3.3.1.11. Source Code

The Source Code field is a two-character designation of the DRGS system the messages was received on. Presently, the following source codes have been defined:

Table 6: DCS Source Codes

Code	Site Source/System
UP	NOAA WCDA E/W Prime – Wallops Island, VA
UB	NOAA WCDA E/W Backup – Wallops Island, VA
NP	NOAA NSOF E/W Prime – Suitland, MD
NB	NOAA NSOF E/W Backup – Suitland, MD
XE	USGS EDDN East – EROS, Sioux Falls, SD
XW	USGS EDDN West – EROS, Sioux Falls, SD
RE	USACE MVR East – Rock Island, IL
RW	USACE MVR West – Rock Island, IL
d1	NIFC West Unit 1 – Boise, ID
d2	NIFC West Unit 2 – Boise, ID
LE	USACE LRD East – Cincinnati, OH
SF	SFWMD East – West Palm Beach, FL
OW	USACE NOW – Omaha, NE

3.3.1.12. Source Secondary

The Secondary Source code is a field that has been suggested as “value-added” source information. Two octets have been reserved for this field, but the specifics of the field are still to be defined.

These fields will be set to two nulls 0x00.

3.3.2. DCP\_DATA := 0..\*(octet)

The DCP\_DATA field is the variable length message data received from the platform or the additional text information for an informational message.

For a DCP message, the data is provided as received; i.e. NO \$ character substitution is performed (e.g. for byte with parity errors).



### 3.4. Missed Message Block

Block type 0x02 designates a missed DCP message block.

A Missed Message block is similar to a DCP Message block, but with unused Header fields omitted and there is no actual DCP data following the header. The Missed Message block consists of the following fields:

MM_ID	Missed Message Block ID (:= 0x02)
BLK_LNG	Missed Message Block Length (see Section 3.2.2)
MM_HDR	Missed Message Header (see Section 3.4.1)
BLK_CRC16	16-bit CRC (see Section 3.2.4)

#### 3.4.1. MM\_HDR

The Missed Message Header is a 24-byte field defined by Table 7.

**Table 7: Missed Message Header**

Field Name	Bytes	Format
Sequence Number	3	Integer Unsigned
Message Flags/Baud	1	Bit Mapped
Platform Address	4	Hexadecimal
Window Start	7	BCD
Window End	7	BCD
Channel/Spacecraft	2	Integer Unsigned/Bit Mapped

Total: 24

#### 3.4.1.1. Missed Message Flags/Baud

The Message Flags field is a bit-mapped byte defined utilized as follows:

**Table 8: DCP Missed Message Flags**

B7	B6	B5	B4	B3	B2	B1	B0
					Data Rate: 001=100; 010=300; 011=1200 100=400; 101=800 000=Undefined; 110-111 Reserved		
					Not Used		
					Reserved for Future		
					Reserved for Future		
					Reserved for Future		
					Reserved for Future		

Unused or reserved bits will be set to 0.

Note that the Data Rate information will be set based on the database definition for the Platform.

#### 3.4.1.2. Platform Address

This Platform Address is a 4-byte hexadecimal (binary) field providing the Platform Address of the Missed Message.



### 3.4.1.3. Window Start

The Window Start is a 7-byte BCD numeric field indicating the start of the self-timed window for the expected, but missed, message. The BCD field format is: YYDDHMMSSZZ where

YY	=	last two digits of the year
DDD	=	Julian day of the year
HH	=	Hour
MM	=	Minute
SS	=	Second
ZZZ	=	Sub-second to millisecond resolution

The byte order is from least significant digit of the sub-seconds (ZZZ) to the most significant digit of the year (YY); i.e. in little endian format.

### 3.4.1.4. Window End

The Window End field is a 7-byte BCD numeric field indicating the end of the self-timed window for the expected message. The BCD field format is the same as the Window Start field defined in the previous section.

### 3.4.1.5. Channel/Spacecraft

The Channel/Spacecraft field provides the DCS channel and satellite messaged was expected on. This field follows the same format as a standard DCP Message (see Section 3.3.1.10) but is filled in based on the anticipated values from the DCP database.

### 3.5. FILE\_CRC32 := 4\*octets

The FILE\_CRC32 is a 32-bit CRC generated per IEEE RFC 1952 on all bytes in the file preceding the FILE\_CRC32 field. Note that this includes the FILE\_HDR and the contained HDR\_CRC32 field in it as well. The value should be stored in the file in little-endian format.

See Section 3.6 for code samples on how to compute the CRC.



### 3.6. DCP Binary Message Blocks

DCP Message blocks are the primary data blocks in the HRIT DCS file. This section defines the format for Binary messages, which also includes messages sent in ASCII or Pseudo Binary compacted binary formats, but were unable to be decompacted due to a CRC error.

DCP Binary Message blocks have a block type identifier of 0x03 and have a variable length based on the DCP message header and DCP data. The DCP Binary Message block consists of the following fields

DCP_ID	DCP Message Block ID (:= 0x03)
BLK_LNG	DCP Message Block Length (see Section 3.2.2)
DCP_B_HDR	DCP Binary Message Header (see Section 3.6.1)
DCP_DATA	DCP Message Data (data as received from DCP or informational message)
BLK_CRC16	16-bit CRC (see Section 3.2.4)

#### 3.6.1. DCP\_B\_HDR

The DCP Binary Message Header is a 36-byte field defined by Table 9. The DCP Binary Message Header is identical to the ASCII and PB Message in both structure and length. The only difference between the two headers is the bit mapping of the Message Flags/Baud field; all other fields are identical to those of an ASCII and Pseudo Binary Message and are defined in the Sections referenced in Table 9.

**Table 9: DCP Binary Message Header**

Field Name	Bytes	Format	Section
Sequence Number	3	Integer Unsigned	3.2.5
Message Flags/Baud	1	Bit Mapped	3.6.1.1
Message ARM Flags	1	Bit Mapped	3.3.1.2
Corrected Address	4	Hexadecimal	3.3.1.3
Carrier Start	7	BCD	3.3.1.4
Message End	7	BCD	3.3.1.5
Signal Strength X10	2	Integer Unsigned	3.3.1.6
Frequency Offset X10	2	Integer Signed	3.3.1.7
Phase Noise X100	2	Integer Unsigned	3.3.1.8
Good Phase X2	1	Integer Unsigned	3.3.1.9
Channel/Spacecraft	2	Integer Unsigned/Bit Mapped	3.3.1.10
Source Code	2	ASCII Characters	3.3.1.11
Source Secondary	2	TBD	3.3.1.12

Total: 36



3.6.1.1. Binary Message Flags/Baud

The Binary Message Flags field is a bit-mapped byte defined utilized as follows:

**Table 10: DCP Binary Message Flags**

B7	B6	B5	B4	B3	B2	B1	B0
					Data Rate: 001=100; 010=300; 011=1200 000=Undefined; 100-111 Reserved		
					1=> Lat/Lon/TxID or DCPC Ack Binary; 0=> Standard		
					1=> Message Received with CRC Errors		
					1=> Binary Message Length Error		
					Binary BCH Flag/Length Field: 1=> Corrected; 0=> Valid		
					Binary BCH Flag/Length Field Uncorrectable		



#### 4. CRC Generation

Disclaimer: Microcom Design Inc. nor the United States Government nor any of their data or content provider shall be liable for any errors in the content of this document, or for any actions taken in reliance thereon. All data and information contained herein is provided for informational purposes only, and is the users responsibility if he/she decides to use or incorporate into their design.

##### 4.1. CRC32 Generation C Code Example

```
/* Generate Table of CRCs for all bytes for a fast CRC */
unsigned long crc32_table[256];
void create_crc32_table(void) {
    unsigned long c;
    int n, k;

    for (n = 0; n < 256; n++) {
        c = (unsigned long) n;
        for (k = 0; k < 8; k++) {
            if (c & 1)
                c = 0xEDB88320L ^ (c >> 1);
            else
                c = c >> 1;
        }
        crc_table[n] = c;
    }
}

/* Update a running crc with the bytes buf[0..len-1] and return
the updated crc. The crc should be initialized to zero. Pre- and
post-conditioning (one's complement) is performed within this
function so it shouldn't be done by the caller. */
unsigned long update_crc(unsigned long crc, unsigned char *buf, int len) {
    unsigned long c = crc ^ 0xFFFFFFFFL;
    int n;

    for (n = 0; n < len; n++)
        c = crc_table[(c ^ buf[n]) & 0xFF] ^ (c >> 8);
    return c ^ 0xFFFFFFFFL;
}
```

##### 4.2. CRC16 Generation C Code Example

```
// CRC-16 Lookup Table
unsigned short crc16_table[] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
    0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
    0x2462, 0x3443, 0x0420, 0x1401, 0x44E6, 0x54C7, 0x64A4, 0x7485,
    0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
    0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
    0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
    0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
}
```



```
0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,  
0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,  
0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,  
0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,  
0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,  
0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,  
0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,  
0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,  
0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,  
0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,  
0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,  
0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,  
0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,  
0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,  
0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,  
0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,  
0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,  
0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,  
0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,  
0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,  
0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0,  
};  
  
// Forward declaration of functions  
unsigned short crc16_gen(const unsigned char *buf, int len);  
  
// Main Body  
.  
.  
.  
unsigned short crc16_gen (const unsigned char *buf, int length)  
{  
    unsigned short crc=0xffff,ind=0;  
  
    while (len != 0) {  
        crc = (crc << 8) ^ crc16_table[(crc>>8)^(unsigned short)buf[ind++]];  
        len--;  
    }  
    return crc;  
}  
  
.  
.  
.  
  
// CRC Calculation Routine  
unsigned short CRC16_checksum;  
  
CRC16_checksum = crc16_gen(dataptr,datalength);
```



## 5. Revision Notes

### 5.1. Revision 1

- 5.1.1. Corrected Frequency Offsetx10 Format in Figure 2 – reserved bits were not shown correctly. Minor correction to corresponding text.
- 5.1.2. Corrected Channel/Spacecraft Format in Figure 4 – was missing B<sub>0</sub>
- 5.1.3. Removed erroneous Received Address section; was 3.3.1.4, Carrier Start is now Section 3.3.1.4.
- 5.1.4. Clarified byte order of Carrier Start and Window Start fields.
- 5.1.5. Updated Section 4 to include disclaimer and CRC16 example. Added C-code syntax coloring.
- 5.1.6. Add No EOT flag bit to Message Flags/Baud in Section 3.3.1.1, Table 2.
- 5.1.7. Added Phase Modulation Index field to Phase Noise field in Section 3.3.1.8, Figure 3, and Table 4.

### 5.2. Revision 2

#### 5.2.1. Clean Up Revisions

- 5.2.1.1. Removed comment about the legacy LRIT/HRIT file format Type code “DCSD” in Section 3.1.4.
- 5.2.1.2. Changed EXP\_FILL and HDR\_CRC32 from level 4 to level 3 outline (i.e. 3.1.4.1 to 3.1.5 and 3.1.4.2 to 3.1.6).
- 5.2.1.3. Added Section 3.2.5 to specific how Sequence Numbers are implemented and can be utilized.
- 5.2.1.4. Added sentence that states unused or reserved bits will be set to 0 in Section 3.3.1.2,

#### 5.2.2. Binary Support Revisions

- 5.2.2.1. Edited Section 3.3 to highlight that what used to be a generic DCP Message Block is now specifically an ASCII and Pseudo Binary Message Block.
- 5.2.2.2. Defined previously Reserved DCP Message Flag Bits (B6 and B7) to support special ASCII and Pseudo Binary Decompacked messages that are part of the Binary DCS message protocol; Table 2 in Section 3.3.1.1.
- 5.2.2.3. Since all DCP Message Flag Bits are defined, removed sentence that stated unused or reserved bits will be set to 0 in Section 3.3.1.1.
- 5.2.2.4. Added new Block Type 0x03 for Binary formatted messages, and Section 3.6 to detail the specifics of the DCP Binary Message Blocks Binary.