Image Probe Data Reference Manual

DOC-0201 Rev B

2545 Central Avenue
Boulder, CO 80301 USA

COPYRIGHT © 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

Copyright © 2011 Droplet Measurement Technologies, Inc.

2545 CENTRAL AVENUE
BOULDER, COLORADO, USA 80301-5727
TEL: +1 (303) 440-5576
FAX: +1 (303) 440-1965
WWW.DROPLETMEASUREMENT.COM

All rights reserved. No part of this document shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from Droplet Measurement Technologies, Inc. Although every precaution has been taken in the preparation of this document, Droplet Measurement Technologies, Inc. assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

Information in this document is subject to change without prior notice in order to improve accuracy, design, and function and does not represent a commitment on the part of the manufacturer. Information furnished in this manual is believed to be accurate and reliable. However, no responsibility is assumed for its use, or any infringements of patents or other rights of third parties, which may result from its use.

Trademark Information

All Droplet Measurement Technologies, Inc. product names and the Droplet Measurement Technologies, Inc. logo are trademarks of Droplet Measurement Technologies, Inc.

All other brands and product names are trademarks or registered trademarks of their respective owners.

CONTENTS

1.0 Introduction 4

2.0 Physical Layer 4

3.0 Sealevel Interface DLL 5

4.0 Communications Parameters 6

5.0 Monoscale Probe Image Format 6

5.1 Compression Algorithm 6
5.2 Format of Uncompressed Data 8
5.2.1 Determining Slice Boundaries 9
5.2.2 Interpreting the Particle Header 10
5.2.3 Run-length Encoder Headers vs. Particle Headers 13
5.2.4 Particle Image Data 13

6.0 Grayscale Probe Image Format 14

6.1 Compression Algorithm 15
6.2 Format of Uncompressed Data 15
6.2.1 Header Slice 16
6.2.2 Image Slices 16
6.2.3 Trailer slices 17

7.0 PADS Raw Data Files 17

7.1 Image Files (ImagefileN_YYYYMMDDHHMMSS) 17
7.1 Index files (ImageindexN_YYYYMMDDHHMMSS)) 18

Appendix A: Source for Sealevel_interface.dll 19

Appendix B: Opening and Closing a SeaMAC Device 23

Appendix C: Mono Image Decompression Example Code 25

Appendix D: Grayscale Example Code 26

C++ Code for Decompression 26
IDL Code for Decompression 28
Fortran Code for Decompression and Parsing 29
IDL Code for Parsing 39

Appendix E: Revisions to Manual 40

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

1.0 Introduction

This manual describes data connection, software interface, data decompression, and storage for mono and grayscale cloud imaging probes (CIPs).

2.0 Physical Layer

See *Appendix B* for information concerning the interface between the Windows API and the Sealevel high speed serial port drivers. These parameters are set by the Open function within the SealevelInterface.dll.

Below is a list of proper configuration parameters for a Sealevel port that can connect to a DMT probe. See *Appendix A* for the SealevelInterface.dll example usage.

```
CommCfg.Electrical = ssiElectricalRS485T;
CommCfg.Framing = ssiFramingSdlc;
CommCfg.CharacterSize = 8;
CommCfg.StopBits = ONESTOPBIT;
CommCfg.PreTxDelayTime = 0;
CommCfg.PostTxDelayTime = 0;
CommCfg.Loopback = FALSE;
CommCfg.Echo = FALSE;
CommCfg.RsetSource = ssiTimingRset;
CommCfg.TsetSource = ssiTimingRset;
CommCfg.TsetFromHere = FALSE;
CommCfg.BitRate = 4000000;
CommCfg.BrgSourceFromRset = FALSE;
CommCfg.CrcPolynomial = ssiCrcCcitt;
CommCfg.CrcPresetOnes = TRUE;
CommCfg.IdleMode = ssiIdleSync;
CommCfg.PreamblePattern = ssiPreamblePatternOnes;
CommCfg.SdlcAddress = 0xFF;
CommCfg.SdlcFlagsShareZero = TRUE;
CommCfg.SyncCharacterSize = 16;
CommCfg.SyncCharacter = 0xFFFF;
CommCfg.DirCon = ssiDirConFullDuplex;
CommCfg.PreambleLength = 0;
CommCfg.ClockEncoding = ssiClockEncodingNone;
CommCfg.SdlcAddressMode = ssiAddressModeFromAny;
CommCfg.PreambleLength = 0;
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
CommCfg.DpllBaseFromRset = FALSE;
CommCfg.MergeFrames = FALSE;
CommCfg.Parity = NOPARITY;
CommCfg.Oscillator = 20000000;
CommCfg.CharacterSyncControl= 0;
dcb.fOutxCtsFlow = FALSE;
dcb.fOutxDsrFlow = FALSE;
dcb.fRtsControl = RTS_CONTROL_ENABLE;
dcb.Parity = NOPARITY;
dcb.BaudRate = CommCfg.BitRate;
dcb.StopBits = ONESTOPBIT;
dcb.ByteSize = 8;
```

3.0 Sealevel Interface DLL

The Sealevel Interface DLL written for the purpose of interfacing DMT imaging probes to Sealevel high speed serial ports.

Functions:

```
Open(* handle, card_number, read_timeout, * error);
Read(* handle, buffersize, * RBuffer, * bytes_read, * error)
Close(*handle)
```

The DLL does not track any memory that is specific to any single probe. It is up to the caller of the DLL to create buffers and variables to pass to each function. The function reads pointers to variables and populates those variables with the appropriate data.

The Open function returns a numeric value based on how far it got in the process of opening the card. If the card was successfully opened, the function returns zero. It populates handle with a handle to the open card. This number is a reference to the read and close functions. Card number is the Sealevel card number given by the driver. Read timeout is the amout of time that is spent (in us) waiting for more data to arrive on the buffer, since the buffer sizes are an expected length, this value is effectively set to a low, non-zero value. Error will contain any Windows API errors thrown in the process of opening the card. Their values can be looked up in the MSDN reference.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

The Read function takes in the handle to the open card, the amount of data that is expected to be read, and the buffer which the caller wants to fill. It populates bytes_read and error.

The Close function takes the handle and closes the card.

Appendix A lists the source code for the Sealevel Interface DLL.

4.0 Communications Parameters

Image data reception requires two receive lines. One is used to receive the data, while the other is used to receive the clock.

Data is sent using a synchronous high-speed serial card. It uses a high-level data link control (HDLC) format.

- Frame Length: 4096 bytes of data, 2 bytes of CRC (cyclic redundancy check)
- Idle state: flags
- CRC: use CRC, preset to ones
- Clock: 3.3 MHz
- RS-422 electrical compatibility

Whenever the image and header data fill a 4096-byte buffer, the probe sends the data to the host software (either PADS or the user's own software). The host software then de-compresses a portion of this data for image display to the screen. It also stores the entire compressed information to disk.

Note: For monoscale probes, there is also an option to send data over a standard asynchronous RS-422 port. This requires a firmware change.

5.0 Monoscale Probe Image Format

5.1 Compression Algorithm

Because image data are large and often contain long strings of ones and zeros, a run-length encoding algorithm is used to compress this information. Compressed data start with a run-length encoding header byte (RLEHB) that looks as follows:

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

Bit 7	Bit 6	Bit 5	Bit 4:0
Z	O	D	Count

Table 1: Format of Run-Length Encoding Header Byte

Z: Zeros Flag. Z = 1 indicates that the encoded bytes to follow the RLEHB are all zeros. After the RLEHB byte, there will be [COUNT+1] bytes that are all zeros.

O: Ones Flag. O = 1 indicates that the encoded bytes to follow the RLEHB are all ones. After the RLEHB byte, there will be [COUNT+1] bytes that are all ones.

D: Dummy Flag. D = 1 indicates that the current RLEHB is a dummy byte and should be ignored. The next byte will be a valid RLEHB. The dummy RLEHB is necessary to ensure that all image buffers begin with a RLEHB.

COUNT: This stores the number of non-header bytes minus one to follow the RLEHB. 0 means one byte to follow, 1 means two bytes to follow, and so on. Non-header bytes can be all zeros, all ones, or unencoded data bytes.

Note: If Z, O, and D are all zero, the next [COUNT + 1] bytes following the RLEHB contain data bytes, or unencoded data.

Example

An example of compressing a string of data will show how the algorithm works.

Before Compression:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
11101111	10010010	00000000	11111111	00000000	00000000	11111111	11111111

Byte 9	Byte 10	Byte 11	Byte 12	Byte 13
11111111	11111111	11001100	11001100	11001100

After Compression (RLEHBs marked in pink):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
00000011	11101111	10010010	00000000	11111111	10000001	01000011	00000010

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

Byte 9	Byte 10	Byte 11
11001100	11001100	11001100

In the example, the first compressed byte is an RLEHB. Since the byte's Z and O flags are both zero, and the byte is not a dummy, it indicates that uncompressed data bytes will follow. In particular, the byte's count value of three means the next four bytes are uncompressed values to be stored. Thus bytes 1-4 in the uncompressed data match bytes 2-5 in the compressed data. (Note that uncompressed bytes 3 and 4 are all zeros and all ones, respectively; they have not been compressed because single bytes of all one value usually are left as data bytes. Ones or zeros are usually compressed using the O and Z flags when there are at least two contiguous bytes of them.)

Bytes 5 and 6 in the uncompressed data are all zeros, and in the compressed data these are converted into an RLEHB in byte 6. This byte has the Z flag high, and [COUNT = 1], so two all-zero bytes follow. The next byte is another RLEHB, with the O flag high, and COUNT = 3, indicating four all-ones bytes follow. The next byte, byte 8, is the last RLEHB. It indicates there are three bytes of data to follow.

Since the encoding algorithm devotes five bits to COUNT, a maximum value of COUNT=31 is allowed. When this occurs, the CIP software starts a new RLEHB.

The system insures data integrity via a CRC checksum. This is handled at the hardware level, so the receiving computer can check for error codes in the hardware card *and no checksum will be required within the data*. A RLEHB begins each 4096-byte image buffer, so even if an error occurs in one image transfer, the receiving software is guaranteed to be able to realign itself to the decompression headers on the next valid received image buffer.

5.2 Format of Uncompressed Data

These data follow the format listed below:

- 64 bits of alternating 1's and 0's (8 bytes of hexadecimal AA's) indicate a particle boundary.
- A particle header, described below, follows each particle boundary.
- Each 64-bit (8-byte) segment of image data is called a "slice." Particle boundaries can be used to determine slice boundaries.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

- In slices that contain actual pixel image data—i.e., slices that are not particle headers or boundaries—1's indicate an illuminated diode on the photodetector, while 0's indicate a shadowed diode and a particle segment.

Note that the compressed data does not necessarily begin on a particle header boundary. It is more likely that the compressed data begins in the middle of a particle, and the first particle boundary must be found by searching for eight bytes of hexadecimal AA (after decompression).

5.2.1 Determining Slice Boundaries

The first step in parsing CIP image data is to determine the slice boundaries. Say the CIP sends the following compressed data:[1]

00 C0 43 00 01 81 00 F0 41 0F AA AA AA AA AA AA AA AA 89 9E 91 AA 3C 66 6C 67 41 03 7F 00 FC FF 43 03

As described earlier, every 4096-byte compressed frame begins with a run-length encoding header byte, so the 00 at the beginning is a RLEHB. It indicates that one byte of non-repeating 0's or 1's follows. The full decompression for the given data is as follows:

C0 FF FF FF FF 01 00 00 F0 FF FF AA AA AA AA AA AA AA AA 89 9E 91 AA 3C 66 6C 67 FF FF 7F 00 FC FF FF FF FF FF .

The eight bytes of AA's in the middle of the data stream indicate a particle boundary and also the slice boundaries for the entire data set:

```
                    C0  FF  FF
FF  FF  01  00  00  F0  FF  FF
AA  AA  AA  AA  AA  AA  AA  AA
89  9E  91  AA  3C  66  6C  67
FF  FF  7F  00  FC  FF  FF  FF
FF  FF
```

The data above has four full slices and two partials. Now the details of the data may be examined.

[1] The CIP sends data in binary format, but a hexadecimal system is used here to preserve space and improve clarity.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

5.2.2 Interpreting the Particle Header

The first slice of each particle image is an image header, which contains a great deal of information about the subsequent particle. The particle header stores the following fields:

- A particle number or count (16 bits)
- A real-time clock time stamp of the time the particle ended (40 bits), beginning with the least significant byte.
- A depth-of-field (DOF) flag (1 bit)
- A slice count (7 bits)

Warning: A key fact in the decompression algorithm is that the data following the alternating 0's and 1's slice is compressed low byte first, high byte second. Therefore, the first number following the last AA (of the eight consecutive AA's) must be assumed the least significant byte of the particle count, and the following number the most significant byte. This wouldn't be mentioned except that it is possible to start a new RLEHB on the high byte of a 16-bit word, so the alignment must all be based on the slice of AA's ending.

The particle header here is the following slice:

89 9E 91 AA 3C 66 6C 67

Figure 1 depicts the particle header's structure. The parameters stored in this header—particle number, time stamp, DOF, and slice count—can then be determined as outlined in the following sections.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

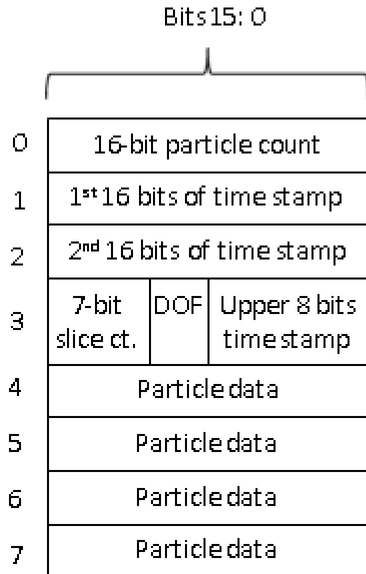

Figure 1: Particle Header Structure

1. Particle Number

The particle number originates from a counter that keeps track of every particle that the CIP hardware sees. It starts at particle 0 and goes up to 65535, after which it rolls over back to a count of 0. When particle images are not recorded, due to inability of the electronics to keep up, the particle number will jump by more than an increment of one, which allows determination of the number of unrecorded particles. For the above example, the particle number is determined from the first two bytes, and is constructed to be 9E89 hex, or 40,585 decimal; the next particle's header should, if no images were lost, begin with 9E8A.

2. Real-time Clock Time Stamp

The real-time clock resolution is 125 nS (clock rate = 8 MHz), so time must be tracked to a resolution of $125 * 10^{-9}$ S. The time stamp stored in the particle header is the time at which the particle ends, with the least significant byte first. To figure out its value in the example header, first rearrange the five bytes after the particle number:

1st (least) 16 bits of time stamp: AA 91
2nd 16 bits of time stamp: 66 3C
last (most significant) byte of time stamp 6C,

6C 66 3C AA 91 hexadecimal =
0110 1100 0110 0110 0011 1100 1010 1010 1001 0001 binary

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

The five most significant bits hold the hour, in 24 hour time format:

0110 1100 0110 0110 0011 1100 1010 1010 1001 0001
01101, or 13, for 1:00 p.m.

The next six most significant bits hold the minute:

0110 1**100 011**0 0110 0011 1100 1010 1010 1001 0001
100011 = 35 minutes

The next six most significant bits hold the seconds:

0110 1100 011**0 0110 0**011 1100 1010 1010 1001 0001
001100 = 12 seconds

The next 10 most significant bits hold the milliseconds:

0110 1100 0110 0110 0**011 1100 101**0 1010 1001 0001
011 1100 101 = 485 milliseconds

And the least 13 significant bits have a resolution of 125 * 10^{-9} S:

0110 1100 0110 0110 0011 1100 101**0 1010 1001 0001**
0 1010 1001 0001 = 2705 (125 * 10^{-9}) = 338.125 μS

3. Slice Count

The slice count is the most significant seven bits of the last byte in the header. The last byte is hexadecimal 67; in binary this is 0110 0111, and the most significant seven bits are 0110011, or 51 decimal. This means there are 51 slices, or samples, of the particle. This count even provides the size of the particle—it is 25μm*$slice^{-1}$ * 51 slices = 1275 μm. (Actually, every particle ends with an alternating ones and zeroes slice, which means the particle is out of the beam, so the size of this particle is reduced to 1250 μm.) The slice count also tells you it is 51 slices * 8 bytes per slice = 408 bytes from the last byte of this header to the first byte of the next.

4. Depth of Field

The depth of field flag follows the slice count, and is active high. This means that if there is a 1 in the DOF bit field, the illumination on at least one diode of the probe's diode array caused the nominal signal level (that which is seen when no particle is present) to drop to 1/3. To record a particle, the nominal signal must drop by ½ or more, so a signal drop to 1/3 means the sampled particle is more centered in the instrument's depth of field. In the example, looking back to the binary representation of 67, we see DOF equals one:

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

0110 011**1**,

so the particle meets depth-of-field requirements for sizing.

5.2.3 Run-length Encoder Headers vs. Particle Headers

The run-length encoding header byte should not be confused with the particle header that has just been discussed. First, the RLEHB contains decompression information, while the particle header contains information about particles. Also, since the RLEHB begins every image frame, it is easy to find; this is not so for the particle headers. Aside from in the first image frame, a particle header can begin anywhere in the data. Thus, if the algorithm for decompressing and analyzing data does not start at the very first particle and work through every single frame, there is a challenge: to find the first particle header in a frame. Two occurrences in the data make this possible. First, every particle header is preceded by an alternating ones and zeros slice in the image (as discussed previously). Second, once the alternating zeros and ones slice is found (eight bytes of hex AA), an algorithm can look at the subsequent particle count, assuming it is a particle header, and then, from the assumed position of the slice count for this particle, jump to the next assumed particle header, checking to see if the particle count has incremented by one. If so, the first header has been found. The only risk is that the perhaps some particles have been lost, so the next particle header has a particle count incremented by more than one. If this occurs, the algorithm would move on to the next slice of alternating zeroes and ones and try again.

5.2.4 Particle Image Data

Assuming a particle header has been found, the subsequent data are purely image recordings. Each slice of a particle's image must be swapped, such that if the bytes are indexed with the first byte received = b[0], and the last byte of the slice received = b[7], like so:

b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7],

then the correctly rearranged bytes for image display or analysis would be as follows:

[b7], b[6], b[5], b[4], b[3], b[2], b[1], b[0].

In image data, a 0 indicates a shadowed diode, i.e. a particle segment, while a 1 indicates an illuminated diode or no detected particle segment.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

Example

The following example illustrates an image data stream that is decompressed and then rearranged in the proper order.

Compressed:
42 00 03 45 03 3F 00 F8 FF 43 03 0F 00 C0 FF 43 03 07 00 80 FF 43 00 01 81 42

Decompressed and arranged in slices:
FF FF FF 03 FF FF FF FF
FF FF 3F 00 F8 FF FF FF
FF FF 0F 00 C0 FF FF FF
FF FF 07 00 80 FF FF FF
FF FF 01 00 00 FF FF FF

Rearranged:
FF FF FF FF 03 FF FF FF
FF FF FF F8 00 3F FF FF
FF FF FF C0 00 0F FF FF
FF FF FF 80 00 07 FF FF
FF FF FF 00 00 01 FF FF
(64 1)

The 64 and the 1 after the block above reference the order of the diodes physically being shadowed. Thus, once the bytes are rearranged, diode 64 is the most significant bit, and diode 1 is the least significant. This corresponds to the numbered test points on the diode array board.

6.0 Grayscale Probe Image Format

Data and index files are recorded in the same manner as monochromatic probes, only the data compression routine differs.

See Appendix D for example code that decompresses and parses CIP-GS image data.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

6.1 Compression Algorithm

To make best use of the nature of the data, the compression scheme treats the data as a continuous stream of 2-bit quantities. Once compressed, the data stream becomes a sequence of bytes. The compressed data consists of two types of bytes: data bytes and count bytes. The bytes are formatted as follows:

Bit	7	6	5	4	3	2	1	0
6-bit	0	1	P0 msb	P0 lsb	P1 msb	P1 lsb	P2 msb	P2 lsb
4-bit	0	0	0	1	P0 msb	P0 lsb	P1 msb	P1 lsb
2-bit	0	0	0	0	0	1	P0 msb	P0 lsb
Count	1	Count of times to repeat last 2-bit data						

2D Data Decompression

The first step in decompressing the Grayscale image data is to look at bit 7 of any byte of the string. This bit will determine if the byte is a count to repeat the last 2 bit data or whether to read a single byte.

Being able to interpret every byte in the compressed data stream allows decompressing to begin anywhere in the stream. To align on a slice boundary, the decompression algorithm needs to look for the 1st 0 after 256 or greater consecutive 1's. This 0 is the 1st bit of a particle header.

Below is a good starting algorithm to grayscale decompression:

- Loop through data, looking at the MSB. If that bit is a 1, repeat the previous byte's data.
- After decompression, search for 128 3's to find the particle boundaries, continue to read 3's until no more (there could be more than 128, but 128 definitely marks the boundary).
- After the end of the 3's, there will be 28 bytes of 0's.
- The next 36 bytes will be the particle header information.
- These header bytes need to be compressed again to obtain the original particle header information. The particle header information should then match header documentation above.

6.2 Format of Uncompressed Data

The image data is composed of 128-bit slices. Every particle is composed of three slice types: a header slice, 1 to 100 image slices, and 2 consecutive trailer slices.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

NOTE: 128-bit header and image slices must be pair-wised reversed during parsing. Bit 1 and bit 2 are exchanged, bit 3 and bit 4 are exchanged, and so on, up to bit 127 and 128. So if your slice starts with

01 10 11 01

This would be changed to

10 01 11 10

You must always determine the first bit of the slice and start pair-reversal with that bit.

6.2.1 Header Slice

After pair-wise reversal (see above), the header slice can then be parsed as follows:

slice[127:120]	8-bit slice count
slice[119:115]	5-bit hours
slice[114:109]	6-bit minutes
slice[108:103]	6-bit seconds
slice[102:93]	10-bit milliseconds
slice[92:83]	10-bit microseconds
slice[82:80]	3-eights of microseconds (125 ns)
slice[79:64]	16-bit particle count
slice[63:56]	8-bit true airspeed (in meters per second)
slice[55:0]	56-bit 0’s

6.2.2 Image Slices

After pair-wise reversal (see above), the image slices are parsed as follows:

slice[127:126] 2-bit value of diode 64 where

- 11 = no shadow
- 10 = 1st level of shadow
- 01 = 2nd level of shadow
- 00 = darkest shadow

slice[125:124] 2-bit value of diode 63**

…

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

slice[1:0] 2-bit value of diode 1

6.2.3 Trailer slices

Trailer slices are 256-bits of consecutive 1's.

7.0 PADS Raw Data Files

The following section is pertinent only to customers who have DMT's Particle Analysis and Display System (PADS).

In PADS, the CIP, CIP-GS, MPS and PIP image file names are formatted as follows:

- In PADS 2.8 and earlier, the file name is ImagefileN_YYYYMMDDHHMMSS, where N-1 = the image port number. If N=1, it is left out so that the filename becomes Imagefile_YYYYMMDDHHMMSS.
- In PADS 3.X, the file name is ImagefileN_YYYYMMDDHHMMSS, where N=the instrument number in PADS + 1.

This file contains raw binary images, along with some time stamp data. Since the image file can become quite large, the data is time stamped, and a separate file, the index, is created that is composed only of timestamps. Once image files become larger than ~80MB, (20,000 writes of 4112 bytes), a new set of image and index files are created with new time stamp in the file name. The index files are named just as the image files are, with a minor exception: instead of Imagefile, image index files are named with ImageindexN_YYYYMMDDHHMMSS

7.1 Image Files (ImagefileN_YYYYMMDDHHMMSS)

PADS image files begin with the following time stamp structure:

```
struct TIME_FIELDS{
    short year;
    short month;
    short day;
    short hour;
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
short minute;
short second;
short milliseconds;
short weekday; };
```

Short data expressions are unsigned, two bytes. The first two bytes are the year. An example of the first 16 bytes of a file is as follows:

D0 07 07 00 06 00 0D 00 23 00 0C 00 71 02 04 00

Note that this data is not compressed. D0 is the first byte of the file, 07 the second, and so on. The year is constructed as 07D0, which, when converted from hexadecimal, is 2000. *Note that all 16 bit values of the TIME_FIELDS structure are byte reversed in the data*. Next, the month is 0007, or July. Then the day is 0006, and 000D for the hour (13, or 1:00 p.m.), 0023 (35 decimal) for the minute, 000C (12) for the second, 0271 (625) for the milliseconds, and 0004 for the day (Thursday—0 is Sunday, 6 is Saturday).

Following the time stamp are 4096 bytes of *compressed* image data from the CIP probe.

7.1 Index files (ImageindexN_YYYYMMDDHHMMSS))

Because the PADS image files can grow so large, an index file is used to simplify the task of finding data that occurs at a specific time in the file. A file is created with the same name as the image file, but with a 2X1 or 2X2 extension. The index files begin with the following time stamp structure:

```
typedef struct
{
   TIME_FIELDS time;
   long offset;
}
```

where TIME_FIELDS is the same structure as defined for the image files, and offset is the offset to the beginning of the matching time field in the image file. An offset value of zero points to the 1st byte of the image file, while noting from above that every image frame takes 4112 bytes, the second image frame in a file would have an offset of 4111 in the index file.

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

Appendix A: Source for Sealevel_interface.dll

```
// Sealevel_interface.cpp : Defines the exported functions for the DLL application.
//This application holds no global storage, it only modifies storage from the caller.
//It is the caller's duty to allocate memory.
//This DLL can be run in many threads concurrently without regard from the caller(s).

#include "stdafx.h"
#include "Ssismv4Deviceinterface.h"
#include <String>
using namespace std;

extern "C" __declspec(dllexport) int Open(HANDLE* handle, int card_number, DWORD
read_timeout, DWORD* error);
extern "C" __declspec(dllexport) int Close(HANDLE* handle);
extern "C" __declspec(dllexport) int Read(HANDLE* handle, DWORD buffersize, char*
RBuffer, DWORD* bytes_read, DWORD* error);

inline string ToString(int x)
{
   char buf[32];
   _itoa_s( x, buf, 32, 10 );
   return string( buf );
}

//Main entry point for DLL
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                             )
{
    return TRUE;
}


//Function: Open- Calls the WinAPI to obtain a handle.  Uses a standard configuration,
// if configuration must be modified, the code below needs to be changed and
recompiled.
//Receives:  Card number, read timeout
//Returns:    Error, non-zero number based on how far it got before error
//                   if no error, returns 0.
__declspec(dllexport) int Open(HANDLE* handle, int card_number, DWORD read_timeout,
DWORD* error)
{

       BOOL bSuccess;
       DCB dcb ;
       DWORD dwTemp ;
       SSI_COMM_CONFIG  CommCfg;
       COMMTIMEOUTS cto ;
//Create a temporary string for the location of the SeaMAC card without the card
number:
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
        string location = "\\\\.\\SeaMAC";
        string location_with_number = location + ToString(card_number);
        //Convert to the legacy LPCSTR type, string to CString to LPCTSTR to LPCSTR
        LPCSTR          location_with_number_LPC                      =(LPCSTR)(LPCTSTR)
location_with_number.c_str();
   *handle = CreateFile(
                location_with_number_LPC,
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                NULL
                );

    if (*handle == INVALID_HANDLE_VALUE) {
        *error = GetLastError() ;
        return (1) ;
    }

   // get initial state of the device's DCB structure
        bSuccess =
                GetCommState(
        *handle,                    // returned from CreateFile()
        &dcb                        // will contain current timeout values
        ) ;

        if (!bSuccess)
        {
                *error = GetLastError() ;
                return 2;
        }

        // get initial state of the device's SSI_COMM_CONFIG structure
        bSuccess =
                DeviceIoControl(
        *handle,                    // returned from CreateFile()
        IOCTL_SSI_GET_COMM_CONFIG,  // I/O control code
        NULL, 0,                    // no input to this function
        &CommCfg, sizeof CommCfg,   // indicate return buffer
        &dwTemp,                    // bytes returned not interesting
        NULL                        // not overlapped (even if opened overlapped)
        ) ;

        if (!bSuccess)
        {
                *error = GetLastError() ;
                return 3;
        }

        CommCfg.Electrical = ssiElectricalRS485T;
        CommCfg.Framing = ssiFramingSdlc;
        CommCfg.CharacterSize = 8;
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
CommCfg.StopBits = ONESTOPBIT;
CommCfg.PreTxDelayTime = 0;
CommCfg.PostTxDelayTime = 0;
CommCfg.Loopback = FALSE;
CommCfg.Echo = FALSE;

CommCfg.RsetSource = ssiTimingRset;
CommCfg.TsetSource = ssiTimingRset;
CommCfg.TsetFromHere = FALSE;
CommCfg.BitRate = 4000000;
CommCfg.BrgSourceFromRset = FALSE;          //note
CommCfg.CrcPolynomial = ssiCrcCcitt;
CommCfg.CrcPresetOnes = TRUE;
CommCfg.IdleMode = ssiIdleSync;
CommCfg.PreamblePattern = ssiPreamblePatternOnes;
CommCfg.SdlcAddress = 0xFF;
CommCfg.SdlcFlagsShareZero = TRUE;
CommCfg.SyncCharacterSize = 16;
CommCfg.SyncCharacter = 0xFFFF;
CommCfg.DirCon = ssiDirConFullDuplex;
CommCfg.PreambleLength = 0;


CommCfg.ClockEncoding = ssiClockEncodingNone;
CommCfg.SdlcAddressMode = ssiAddressModeFromAny;
CommCfg.PreambleLength = 0;
CommCfg.DpllBaseFromRset = FALSE;
CommCfg.MergeFrames = FALSE;
CommCfg.Parity = NOPARITY;
CommCfg.Oscillator = 20000000;

CommCfg.CharacterSyncControl= 0;

dcb.fOutxCtsFlow = FALSE;          //note
dcb.fOutxDsrFlow = FALSE;                  // don't need DSR to send
dcb.fRtsControl = RTS_CONTROL_ENABLE;      // allow control of RTS
dcb.Parity = NOPARITY;                     // no parity in this application
dcb.BaudRate = CommCfg.BitRate;                      // make it the same as above
dcb.StopBits = ONESTOPBIT;
dcb.ByteSize = 8;

bSuccess =
       DeviceIoControl(
 *handle,                   // returned from CreateFile()
 IOCTL_SSI_SET_COMM_CONFIG, // I/O control code
 &CommCfg, sizeof CommCfg,  // indicate address of structure
 NULL, 0,                   // no output from this function
 &dwTemp,                   // bytes returned not interesting
 NULL                       // not overlapped (even if opened overlapped)
 ) ;

if (!bSuccess)
{
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
		*error = GetLastError() ;

		return 4;
	}
	// update the device's DCB
	bSuccess =
		SetCommState(
	 *handle,			// returned from CreateFile()
	 &dcb				// new values
	 ) ;

	if (!bSuccess)
	{
		*error = GetLastError() ;
		return 5;
	}

	// update the device's SSI_COMM_CONFIG structure
	// get initial timeouts settings before changing them
	bSuccess =
		GetCommTimeouts(
	 *handle,			// returned from CreateFile()
	 &cto				// will contain current timeout values
	 ) ;

	if (!bSuccess)
	{
		*error = GetLastError() ;
		//error_output.Format("Error %u from GetCommTimeouts\r\n", dwLastError);

		return 6;
	}

	// change time out
	cto.WriteTotalTimeoutMultiplier = 0 ;	// no per-character timeout
	cto.WriteTotalTimeoutConstant = 20 ;	// wait a total
	cto.ReadIntervalTimeout = 0 ;		// no interval timeout
	cto.ReadTotalTimeoutMultiplier = 0 ;	// no per-character timeout
	cto.ReadTotalTimeoutConstant = read_timeout ;	// wait a total

	bSuccess =
		SetCommTimeouts(
	 *handle,			// returned from CreateFile()
	 &cto				// new timeout values
	 ) ;

	if (!bSuccess)
	{
		*error = GetLastError();
		return 7;
	}

	*error=0;//no errors happened!
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
	return 0;//Success, the caller should now have a valid handle to an open card.
}
__declspec(dllexport) int Read(HANDLE* handle, DWORD buffersize, char* RBuffer, DWORD* bytes_read, DWORD* error)
{

	BOOL bSuccess;
	bSuccess =
			ReadFile(
			*handle,		// handle
			RBuffer,		// will contain the received data
			buffersize,		// size of passed buffer
			bytes_read,		// actual amount of returned data, in bytes
			NULL			// non-overlapped read
			) ;
	if(!bSuccess)
		*error = GetLastError();
	else
		*error=0;

	return(!bSuccess);

}
__declspec(dllexport) int Close(HANDLE* handle)
{
	BOOL bSuccess;
	bSuccess=CloseHandle(*handle);
	return(!bSuccess);

}
```

Appendix B: Opening and Closing a SeaMAC Device

[Taken verbatim from SeaMAC v4.2 help file]

You can use the CreateFile function to open a SeaMAC device. The device name specifies which device by its zero-based unit number.
In the following example, CreateFile opens SeaMAC device unit zero (0) for normal receiving and transmitting.

```
HANDLE hDevice;

hDevice = CreateFile(
	"\\\\.\\SeaMAC0",	// open SeaMAC unit 0
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
            GENERIC_READ | GENERIC_WRITE,    // open for sending and receiving
            0,                                                // do not share
            NULL,                                             // no security
            OPEN_EXISTING,                                 // device exists
            FILE_ATTRIBUTE_NORMAL,                         // normal access
            NULL);                                       // no attr. template

if (hDevice == INVALID_HANDLE_VALUE)
{
    ErrorHandler("Could not open SeaMAC device");      // process error
}
```

In this example, CreateFile only succeeds if a SeaMAC unit zero (0) device exists. An application should check the return value of CreateFile before attempting to use the handle to access the device. If an error occurs, the application should use the GetLastError function to get extended error information and respond accordingly.

A SeaMAC device must be closed with CloseHandle before it can be re-opened and accessed by another process. When running in a legacy environment and using certain Win32 Serial functions, the application must not call CloseHandle, but must call ssi_CloseHandle instead. The syntax and semantics of ssi_CloseHandle are otherwise identical to CloseHandle.

The following line closes the “SeaMAC0” device:

```
CloseHandle(hDevice) ;
```

The following line closes the “SeaMAC0” device for an application running under the Windows® Me operating system:

```
ssi_CloseHandle(hDevice) ;
```

In the following example, CreateFile opens SeaMAC device unit zero (0) for “overlapped” receiving and transmitting.

```
HANDLE hDevice ;

hDevice = CreateFile(
            "\\\\.\\SeaMAC0",                        // open SeaMAC unit 0
            GENERIC_READ | GENERIC_WRITE,    // open for sending and receiving
            0,                                                // do not share
            NULL,                                             // no security
            OPEN_EXISTING,                                 // device exists
            FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, // normal access with
overlapped                                                             I/O
            NULL);                                       // no attr. template

if (hDevice == INVALID_HANDLE_VALUE)
{
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
    ErrorHandler("Could  not  open  SeaMAC  device");      //  process  error
}
```

Appendix C: Mono Image Decompression Example Code

```
__declspec(dllexport) long decompress_2d(PUCHAR bytes, long bytecount, PUCHAR
uncompressed_bytes, long max_uncompressed_bytes, long *position)
{
    long cur_in = 0;
    long cur_out = 0;
    unsigned char a_byte;
    unsigned char iterator = 0;
    long idx;

        long bad_msg = 0;
    long use_a_byte = 0;
        long uncompressed_bytecount = 0;

        // The first byte is always the index of the first RLE header byte
        cur_in = 0;

    while (cur_in < bytecount) {
        if (cur_in > bytecount) {
            bad_msg = 1;
            break;  // Want out of While
        }
        use_a_byte = 1;

                if ((bytes[cur_in] & 0xFF) & 0x20) {
                        cur_in++;
                        continue;
                }
                else if ((bytes[cur_in] & 0xFF) > 0x7F) {   // > 127
            a_byte = 0x00;
            //iterator = bytes[cur_in] - 128;
            iterator = bytes[cur_in] & 0x7F;
                        cur_in++;
                        }
        else if ((bytes[cur_in] & 0xFF) > 0x3F) {  // > 63
            a_byte = 0xFF;
            //iterator = (bytes[cur_in] - 64) & 0xFF;
            iterator = bytes[cur_in] & 0xBF;
                        cur_in++;
                        }
        else {
            iterator = bytes[cur_in] & 0xFF;
            use_a_byte = 0;
            cur_in++;
            }

        if ((iterator & 0xFF) > 0x1F) {     // > 31
            //NiceMsgBox "Decompress:Iterator > 31 at " & cur_in - 1
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                    *position = cur_in;
          return (-31);  // Exit Sub
                    }
        if ((cur_out + iterator) >= max_uncompressed_bytes)
              {
                    *position = cur_in;
                    return (uncompressed_bytecount);
              }

        //For idx = iterator To 0 Step -1
        for (idx = iterator ; idx >= 0 ; idx--)
                    {
                    if (cur_out >= max_uncompressed_bytes) {
                           // uncompressed_bytes buffer is full
                           uncompressed_bytecount = cur_out - 1;
                           *position = cur_in;
                           return (uncompressed_bytecount);
                    }
            if (use_a_byte) {
                uncompressed_bytes[cur_out] = a_byte;
                cur_out++;
                           }
                    else {
                if (cur_in > bytecount) {
                    bad_msg = 1;
                    break;
                }
                uncompressed_bytes[cur_out] = bytes[cur_in];
                cur_in++;
                cur_out++;
            }
              }
    }

    if (bad_msg) {
              *position = cur_in;
        return (-1);
       }

    uncompressed_bytecount = cur_out - 1;
    *position = cur_in;
    return (uncompressed_bytecount);
}
```

Appendix D: Grayscale Example Code

C++ Code for Decompression

```
__declspec(dllexport) long decompress_Grey(PUCHAR bytes_in, long bytecount, PUCHAR
uncompressed_bytes, long max_uncompressed_bytes, long *position)
{
       // bytes_in is an array of compressed bytes
       // bytecount is the length of the array
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
        // uncompressed_bytes contains the uncompressed image data.  The uncompressed
data is an array of characters representing a 0,1,2,3
        // max_uncompressed contains the length of the uncompressed bytes


        long cur_in = 0;
    long cur_out = 0;
        long count;
    unsigned char Last2bits;
    int i = 0;  // Counter for While loop.  also keep track of position in bytes_in
        int j, counter;

        counter = 0;
        *position = 25;
        Last2bits = 0;

        while (i < bytecount)
        {

                if (bytes_in[i] && 0x80) // check to see if the 8th bit of the current
byte is true or 1
                {
                        uncompressed_bytes[counter]  =  bytes_in[i]&  0x30;    //read  in
bits[4:5] of the current byte convert to #
                        counter++;
                        uncompressed_bytes[counter]  =  bytes_in[i]&  0xC;    //read  in
bits[2:3]
                        counter++;
                        uncompressed_bytes[counter]  =  bytes_in[i]&  0x3;    //read  in
bits[0:1]

                        counter++;
                        Last2bits = bytes_in[i] & 0x3;

                }
                else if ((bytes_in[i] && 0x80) == 0) // check to see if the 8th bit of
the current byte is false or 0
                {
                        count = (bytes_in[i] & 0x7F);

                        for(j=0 ; j<=count; j++)
                        {
                                uncompressed_bytes[counter] = Last2bits;
                                counter++;
                        }
                }
                else
                {
                        return 13;
                }
                i++;

        }
        max_uncompressed_bytes = counter;
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
        return 0;

}

        //  Search for 128 3's to find the particle boundaries
        //  continue to read 3's until no more (there could be more than 128, but 128 
definately marks the boundary)
        //  After the end of the 3's there will be 28 bytes of 0's
        //  The next 36 bytes will be the particle header information
        //  These need to be compressed again to obtain the original particle header 
information
        //  The particle header information should then match original documentation
```

IDL Code for Decompression

DMT gratefully acknowledges UCAR for providing the code below.

```
;Decompress a 4096 byte grey-scale image buffer (cimage)
    im=bytarr(64,2000)   ;The uncompressed buffer, up to 2000 slices
    next=0l
    pairs=bytarr(4)
    repcount=0
    blankdetect=0
    partcount=0
    timeline=lonarr(300)
    FOR i=0,n_elements(cimage)-1 DO BEGIN
       IF cimage[i] eq 255 THEN blankdetect=1
       IF cimage[i] ge 128 THEN BEGIN
          ;This is a "count" byte
          n=cimage[i] and 127b
          im[next:next+n-1]=pairs[3]
          repcount=repcount+n
       ENDIF ELSE BEGIN
          repcount=0
          n=0
          IF blankdetect THEN BEGIN
             next=next-(next mod 64)
;align on an edge boundary, it sometimes drifts...
             timeline[partcount]=next
             partcount=partcount+1
          ENDIF
          blankdetect=0
          pairs=[(cimage[i]  and  64b)/64,  (cimage[i]  and  48b)/16, 
(cimage[i] and 12b)/4, cimage[i] and 3b]
          ;This is a data byte
          IF pairs[2] THEN BEGIN
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
             im[next]=pairs[3]
             n=1
          ENDIF
          IF pairs[1] THEN BEGIN
             im[next]=pairs[2]
             im[next+1]=pairs[3]
             n=2
          ENDIF
          IF pairs[0] THEN BEGIN
             im[next]=pairs[1]
             im[next+1]=pairs[2]
             im[next+2]=pairs[3]
             n=3
          ENDIF
       ENDELSE
       IF repcount gt 256 THEN BEGIN
        ;  print,repcount
          return,{particle_count:0,bitimage:bytarr(64,1000),sync_ind:0,
time_elap:0, time_sfm:0, slice_count:0}
;indicates a bad buffer, exit now
       ENDIF
       next=next+n
    ENDFOR
```

Fortran Code for Decompression and Parsing

```
      program analyzeICE_LGray
c     Created:     January 27, 2008
c
c     This program processes CIP-Gray records
c     This version outputs size spectra every record and PBP per record
      character*50  filein(500)
      character*12 filenam,imageName
      character*17 prefix
      character*8 fltdate
      character*8   fltdat(7)
      byte bufHeader(16),id
      byte   ucmprec(600000),image(64,500),word(16)
      double precision secs(500),seconds,secs0,secsBefore,elapsd
      real cipSize(64),cipSpec(4,64),avg(13),v(13),cf(3)
      real sizStats(3,2,700),frac(3,500),fractal(3),surface
      integer dates(8),slices(500),pcount,pcntlst,fcnt,slcnt
      integer length(3),width(3),area(3),perimeter(3),ndrjct(3)
      integer images(640,500),timeOffset,prtpntr(1000)
      integer w(3,500),l(3,500),a(2,3,500),p(3,500),nsizStats(700)
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
      integer rej(3,500),areaImage(3),areaMax(3),diameterMax
      data    nflts/7/
      data prefix/'/data/C130/ICE-L/'/
      data sizStats/4200*0/,nsizStats/700*0/
      data navg/1.0/,pi/3.14159/
      data    armdst/100/,tas/140/,taur/0.35/
c..........This is the mininum number of pixels shadowed to allow an image to be
c..........stored in the output data file
      data minArea/500/imageFlg/1/
      equivalence    (dates(1),        iyr),
   $                 (dates(2),        imonth),
   $                 (dates(3),        iday),
   $                 (dates(4),        khr),
   $                 (dates(5),        kmin),
   $                 (dates(6),        ksec),
   $                 (dates(7),        kmsec)
c     Time between particle velocity clock, in us
      data clktim/0.5/
c     Set up the I/O paths and averaging times
      write(6,8000)
8000  format('Enter flight date (yyyymmdd) ',$)
      read(5,8001)fltdate
8001  format(a8)
      do ifl=1,nflts
            if (fltdate .eq. fltdat(ifl))exit
      enddo
      open(30,file=prefix//fltdate//'/'//fltdate//
   $'CIPpbpNumberbyRecord.dat')
      write(30,9888)
9888  format('start time End Time Number of particles'/
   $' l1 w1 a1 p1 cf1 l2 w2 a2 p2 cf2 l3 w3 a3 p3 cf3')
c     Read in the filenames
      eof=0
      ifl=0
      open(20,file=prefix//fltdate//'/directory.txt')
      do while (eof.eq.0)
            ifl=ifl+1
            read(20,8005,end=5)filein(ifl)
8005        format(a50)
      enddo
5     close(20)
      nfl=ifl-1
      do ifl=1,nfl
      write(6,8010)filein(ifl)
8010  format(a50)
      open(20,file=prefix//fltdate//'/'//trim(adjustl(filein(ifl))),
   $form='binary',recordtype='STREAM',mode='READ')
      eofflg=0
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
        nparticles=0
        npart=0
        nfrc1=0
        nfrc2=0
        do i=1,13
                avg(i)=0
        enddo
        do i=1,4
                do j=1,60
                        cipSpec(i,j)=0
                enddo
        enddo
        do i=1,640
                do j=1,500
                        images(i,j)=0
                enddo
        enddo
        imageflg=0
        frstFlag=5
        call decompress(ucmprec,bufHeader,nbits,eofflg,frstFlag)
        call decompress(ucmprec,bufHeader,nbits,eofflg,frstFlag)
        do i=1,8
                j= 1 + (i-1)*2
                dates(i) = neword(bufHeader(j),bufHeader(j+1))
        enddo
        secs0=khr*3600 + kmin*60 + ksec + float(kmsec)*.001
        firsttime=secs0
        nrecords=0
        np=0
        nlow=0
        nhigh=0
        dtavg=0
        dtstd=0
        frstFlag=1
        do while (eofflg.eq.0)
c
c       Read a 2D record and decompress it
                call decompress(ucmprec,bufHeader,nbits,eofflg,frstFlag)
c       decode the date and time of this record
c       (this is when the record was written, not the start time)
c       Reference all the particle times to this time, as it
c       is clock time, not the time from probe turn on.
                startm=khr*3600 + kmin*60 + ksec
                if (eofflg.eq.0 .and. nbits.gt.0)then
                        nrecords=nrecords+1
                        do i=1,8
                                j= 1 + (i-1)*2
                                dates(i) = neword(bufHeader(j),bufHeader(j+1))
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                        enddo
                        endtm=khr*3600 + kmin*60 + ksec
                        iptr = 0
                        nslice=nbits/128
c                       do i=1,nslice
c                               write(88,9765)(ucmprec(j),j=(i-1)*128+1,i*128)
9765                            format(128i1)
c                       enddo
c
c       Sort through matrix of slices and locate all the particles.
c       We are going to assume that the first slice in this matrix contains a
c       valid particle header.
c       We search for the last "1", assume that this is the last bit of the
c       two trailing slices, then work backwards, slice by slice,looking for the 56 '0's in
c       a row in order locate the header.
                        iprt=0
                        lsthdr=0
                        do while (iptr .le. nbits)
c
c       Search for 256 or more '1's in a row
                                idcd = 0
                                iflg=0
                                do while (iflg.eq.0 .and. iptr.le.nbits)
                                        iptr=iptr+1
                                        idcd = idcd+ucmprec(iptr)
                                        if (ucmprec(iptr) .eq.0)then
                                                if (idcd.ge.256)then
                                                        iflg=1
                                                end if
                                                idcd=0
                                        end if
                                enddo
                                if (iptr.le.nbits)then
                                        iprt=iprt+1
                                        prtPntr(iprt)=iptr
                                end if
                        enddo
                        npart=iprt
c       Go through the buffer reading the particles
                        ipart=0
                        do iprt=1,npart-1
c       The end of a particle has been found. Put the next 128 bits into
c       a word that will be decoded as the header. The bytes need
c       reordering, however, to take care of Endian problems
                                iptr=prtPntr(iprt)
c       Decode the header
                                call dcdhdr(ucmprec,iptr,ihr,imin,isec,ims,micro,nano,
   $                    slcnt,pcount,tas)
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                        iptr=iptr+128
                        slices(iprt)=(prtPntr(iprt+1)-prtPntr(iprt))/128-2
                        slcnt=slices(iprt)
c     And use the slice count to put the next 128*slcnt into the image buffer
                        do j=1,slcnt
                              do i=1,64
                                    image(i,j)=ucmprec(iptr)*2+ucmprec(iptr+1)
                                    iptr=iptr+2
                              enddo
                        enddo
c                       write(88,9776)(word(i),i=1,16),khr,kmin,ksec,kmsec,
c    $            ihr,imin,isec,ims,micro,nano,slcnt,pcount
c9776                   format(16B8/4i6/8i6)
c                       write(88,9776)khr,kmin,ksec,ihr,imin,isec,ims,slcnt,
c    $            pcount
9776                    format(6i4,i5,i4,i6)
c                       do j=1,slcnt
c                             write(6,9777)(image(i,j),i=1,64)
c                       enddo
9777                    format(64i1)
c     Process this information and the subsequent particle image.
c     Don't do anything if this particle has a zero pcount
                        if (slcnt.gt.0)then
c
c     Process the image to get the maximum width, length, area and
c     anything else of interest about the particle of the particle
                              call size2d(image,iptr,slcnt,width,length,
   $                    areaImage,areaMax,perimeter,fractal,ndrjct)
                              prtFlg=0
                              do i=1,3
                                    if (width(i).gt.0)then
                                          ratio=length(i)/width(i)
                                    else
                                          ratio=20
                                    end if
                                    if (ratio.lt.5 .and.
   $                          areaImage(i).ge.1)prtFlg=1
                              enddo
                              if (prtFlg.gt.0)then
                                    ipart=ipart+1
                                    do i=1,3
                                          w(i,ipart)=width(i)
                                          l(i,ipart)=length(i)
                                          a(1,i,ipart)=areaImage(i)
                                          a(2,i,ipart)=areaMax(i)
                                          frac(i,ipart)=fractal(i)
                                          rej(i,ipart)=ndrjct(i)
                                          p(i,ipart)=perimeter(i)
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                                                enddo
                                        end if
                                end if
                        end do
                end if
                write(30,9950)int(startm),int(endtm),ipart
9950            format(i5,1x,i5,i5)
                do j=1,ipart
                        do i=1,3
c       Correct for the out of focus particles
                                aImage=a(1,i,j)
                                aMax=a(2,i,j)
                                diameterMax=w(i,j)
                                correction=1.0
                                if (aImage .lt. aMax)
    $                           call poisson(areaImage,areaMax,diameterMax,
    $                           correction,Zd)
                                cf(i)=correction
                        enddo
                        write(30,9955)(l(i,j),w(i,j),a(2,i,j),p(i,j),cf(i),i=1,3)
9955                    format(3(i3,i3,i4,i4,f5.2))
                enddo
                nparticles=nparticles+ipart
                if (mod(nrecords,1000).eq.0)print *,nrecords
        end do
        print *,'Total # of nrecords=',nrecords,' Total particles=',
    $nparticles
        enddo
        close(20)
        close(30)
        end
        subroutine decompress(newrec,bufHeader,nslice,eofflg,frstFlag)
        integer*1       nibble,nibble1,nibble2,nibble3,nibble4
        byte newrec(600000),rawrec(4124),bufHeader(16)
        byte word(128),code1,code2,code3,code4
c       Read a record
        eofflg=0
        nread=frstFlag
        do j=1,nread
                read(20,end=9900)(rawrec(i),i=1,4112)
        enddo
        frstFlag=1
c       do i=1,4124
c               write(55,8888)i,rawrec(i)
8888            format(i4,1x,o4)
c       enddo
c       The first sixteen bytes go directly into the new record buffer
        do 5 nslice=1,16
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                bufHeader(nslice)=rawrec(nslice)
5       continue
        iptr=17
        nslice=1
        nibble=0
        do while (iptr.le. 4112)
c       Check if this a count byte or data byte
                code1=iand(rawrec(iptr),2#10000000)
                code2=iand(rawrec(iptr),2#01000000)
                code3=iand(rawrec(iptr),2#00010000)
                code4=iand(rawrec(iptr),2#00000100)
                if (code1.ne.0)then
                        ncnt=iand(rawrec(iptr),2#01111111)
                        do i=1,ncnt
                                newrec(nslice)=iand(nibble,2#00000010)/2
                                newrec(nslice+1)=iand(nibble,2#00000001)
                                nslice=nslice+2
                        enddo
                else
                        if (code2.ne.0)then
                                newrec(nslice)  =iand(rawrec(iptr),2#00100000)/32
                                newrec(nslice+1)=iand(rawrec(iptr),2#00010000)/16
                                newrec(nslice+2)=iand(rawrec(iptr),2#00001000)/8
                                newrec(nslice+3)=iand(rawrec(iptr),2#00000100)/4
                                newrec(nslice+4)=iand(rawrec(iptr),2#00000010)/2
                                newrec(nslice+5)=iand(rawrec(iptr),2#00000001)
                                nslice=nslice+6
                                nibble=iand(rawrec(iptr),2#00000011)
                        else
                                if (code3.ne.0)then
                                        newrec(nslice)=iand(rawrec(iptr),2#00001000)/8
                                        newrec(nslice+1)=iand(rawrec(iptr),2#00000100)/4
                                        newrec(nslice+2)=iand(rawrec(iptr),2#00000010)/2
                                        newrec(nslice+3)=iand(rawrec(iptr),2#00000001)
                                        nslice=nslice+4
                                        nibble=iand(rawrec(iptr),2#00000011)
                                else
                                        if (code4.ne.0)then

        newrec(nslice)=iand(rawrec(iptr),2#00000010)/2

        newrec(nslice+1)=iand(rawrec(iptr),2#00000001)
                                                nslice=nslice+2
                                                nibble=iand(rawrec(iptr),2#00000011)
                                        end if
                                end if
                        end if
                end if
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
              iptr=iptr+1
        end do
        return
9900    eofflg=1
        return
        end
        integer function neword(frst,secnd)
        byte frst,secnd
        integer id1,id2
        id1 = secnd*256
        id2 = frst
        id2 = iand(id2,16#000000FF)
        id1 = iand(id1,16#0000FF00)
        neword = id1+id2
        return
        end
        subroutine dcdhdr(hdr,ihdr,ihr,imin,isec,ims,micro,nano,
   $slcnt,pcount,tas)
        integer pcount,slcnt
        byte    hdr(600000)
        slcnt=hdr(ihdr+127)*128+hdr(ihdr+126)*64+hdr(ihdr+125)*32+
   $hdr(ihdr+124)*16+hdr(ihdr+123)*8+hdr(ihdr+122)*4+hdr(ihdr+121)*2+
   $hdr(ihdr+120)
        slcnt=4+slcnt/2
        ihr=hdr(ihdr+119)*16+hdr(ihdr+118)*8+hdr(ihdr+117)*4+
   $hdr(ihdr+116)*2+hdr(ihdr+115)
        imin=hdr(ihdr+114)*32+hdr(ihdr+113)*16+hdr(ihdr+112)*8+
   $hdr(ihdr+111)*4+hdr(ihdr+110)*2+hdr(ihdr+109)
        isec=hdr(ihdr+108)*32+hdr(ihdr+107)*16+hdr(ihdr+106)*8+
   $hdr(ihdr+105)*4+hdr(ihdr+104)*2+hdr(ihdr+103)
        ims=hdr(ihdr+102)*512+hdr(ihdr+101)*236+hdr(ihdr+100)*128+
   $hdr(ihdr+99)*64+hdr(ihdr+98)*32+hdr(ihdr+97)*16+hdr(ihdr+96)*8+
   $hdr(ihdr+95)*4+hdr(ihdr+94)*2+hdr(ihdr+93)
        micro=hdr(ihdr+92)*512+hdr(ihdr+91)*236+hdr(ihdr+90)*128+
   $hdr(ihdr+89)*64+hdr(ihdr+88)*32+hdr(ihdr+87)*16+hdr(ihdr+86)*8+
   $hdr(ihdr+85)*4+hdr(ihdr+84)*2+hdr(ihdr+83)
        nano=hdr(ihdr+82)*4+hdr(ihdr+81)*2+hdr(ihdr+80)
        pcount=hdr(ihdr+79)*32768+hdr(ihdr+78)*16384+hdr(ihdr+77)*8192+
   $hdr(ihdr+76)*4096+hdr(ihdr+75)*2048+hdr(ihdr+74)*1024+
   $hdr(ihdr+73)*512+hdr(ihdr+72)*236+hdr(ihdr+71)*128+
   $hdr(ihdr+70)*64+hdr(ihdr+69)*32+hdr(ihdr+68)*16+hdr(ihdr+67)*8+
   $hdr(ihdr+66)*4+hdr(ihdr+65)*2+hdr(ihdr+64)
        tas=hdr(ihdr+63)*128+hdr(ihdr+62)*64+hdr(ihdr+61)*32+
   $hdr(ihdr+60)*16+hdr(ihdr+59)*8+hdr(ihdr+58)*4+hdr(ihdr+57)*2+
   $hdr(ihdr+56)

c       Decode the particle count, slice count and time
c       of the particle. If pcount is zero, don't bother to decode
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
c          anything else as this is a bad particle
           return
           end
           subroutine size2d(image,iptr,nslice,width,length,
    $areaImage,areamax,perimeter,fractal,ndrjct)
           real      fractal(3)
           integer width(3),area(3),perimeter(3),ndrjct(3),length(3),on
           integer p,perimeter1,areamax(3),areaImage(3)
           byte      image(64,500)
           do lvl=1,3
                imax=0
                imin=64
                ndrjct(lvl)=0
                areamax(lvl) = 0
                areaImage(lvl)=0
                length(lvl)=0
                width(lvl)=0
                if (nslice.gt.1)then
                     do i=1,nslice
                          ix = 1
                          jmax=0
                          jmin=0
                          do  j=1,64
                               if (image(j,i).lt. lvl)then
                                    if (jmin.eq.0)jmin=j
c          A 0 bit corresponds to a shadowed diode. Set this bit in the image
c          array.
                                    jmax=j
                                    areaImage(lvl)=areaImage(lvl)+1
                               end if
                               ix=ix+1
                          enddo
                          if (jmin.gt.0)length(lvl)=length(lvl)+1
                          if (jmax.ne.0 .or. jmin .ne.0)areaMax(lvl)=areaMax(lvl)+
    $                abs(jmax-jmin)+1
                          if (jmax.gt.0)imax = max0(imax,jmax)
                          if (jmin.gt.0)imin = min0(imin,jmin)
                     end do
                     if (imax.gt.0 .and. imin.gt.0)width(lvl)=abs(imax-imin) + 1
                     if (imin.eq.1 .or. imax.eq.64)then
                          ndrjct(lvl)=1
                     end if
                     if (imin.eq.1 .and. imax.eq.64)then
                          ndrjct(lvl)=2
                     end if
                     icol=64
                     irow=nslice
c                    calculate the perimeter at 25 um resolution
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
                    call perim(icol,irow,image,p,lvl)
                    perimeter(lvl)=p
                end if
        end do
        return
        end
c       This subroutine calculates the image perimeter
        subroutine perim(icol,irow,image,perimeter,lvl)
        integer perimeter
        byte    image(64,500)
c       Scan the image to calculate the perimeter. This calculate all
c       the changes from 0's to 1's in both directions in the array
        perimeter=0
        do i=1,irow
                on=0
                do j=1,icol
                        if (image(j,i).lt. lvl)then
                                if (on.eq.0)then
                                        on=1
                                        perimeter=perimeter+1
                                end if
                        else
                                if (on.eq.1)then
                                        on=0
                                        perimeter=perimeter+1
                                end if
                        end if
                enddo
                if (on.eq.1)perimeter=perimeter+1
        enddo
        do j=1,icol
                on=0
                do i=1,irow
                        if (image(j,i).lt. lvl)then
                                if (on.eq.0)then
                                        on=1
                                        perimeter=perimeter+1
                                end if
                        else
                                if (on.eq.1)then
                                        on=0
                                        perimeter=perimeter+1
                                end if
                        end if
                enddo
                if (on.eq.1)perimeter=perimeter+1
        enddo
        return
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
      end
      subroutine poisson(areaImage,areaMax,diameterMax,
$correction,Zd)
      real image2MaxRatios(17),ZeeD(17),diameterRatios(17)
      integer areaImage,areaMax,diameterMax
      data    image2MaxRatios/
$1.00,0.95,0.90,0.85,0.80,0.75,0.70,0.65,0.60,0.55,0.51,0.45,0.41,
$0.37,0.33,0.29,0.17/
      data    diameterRatios/
$1.00,1.07,1.35,1.61,1.76,1.84,1.88,1.90,1.91,1.90,1.89,1.88,1.87,
$1.86,1.85,1.84,1.81/
      data    Zeed/
$0.02,1.94,3.24,4.58,5.66,6.40,6.92,7.32,7.60,7.80,7.92,8.02,8.08,
$8.12,8.14,8.16,8.18/
      correction = -1
      Zd = -1
      if (areaImage.gt.0 .and. areaMax.gt.0)then
              ratio=sqrt(float(areaImage)/float(areaMax))
              do i=2,17
                      if (ratio .ge. image2MaxRatios(i))exit
              enddo
              iptr=i-1
              Zd = zeed(iptr)
              correction=diameterRatios(iptr)
      end if
      return
      end
```

IDL Code for Parsing

Below is an IDL code excerpt from a grayscale timeline decoding program. DMT gratefully acknowledges UCAR for providing this code.

The call to bin2dec is a simple routine that converts a series of 1s and 0s to a decimal number. "ind" is an increasing integer array 0,1,2...63. Note that IDL indexing starts at 0.

```
    ;Raw header is in 'grey' mode... 64 values from 0 to 3.  Convert to
128-bit slice:
    ind=indgen(64)
    fullslice=bytarr(128)
    fullslice[ind*2+1]=(header and 2b)/2
    fullslice[ind*2]=header and 1b

    ;From 128-bit slice:
    tas=bin2dec(reverse(fullslice[56:63]))
```


© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.

```
    particle_count=bin2dec(reverse(fullslice[64:79]))
    counter=bin2dec(reverse(fullslice[80:82]))  ;  this  is  in  125  ns
increments (8MHz for all probes)
    microsecond=bin2dec(reverse(fullslice[83:92]))
    millisecond=bin2dec(reverse(fullslice[93:102]))
    second=bin2dec(reverse(fullslice[103:108]))
    minute=bin2dec(reverse(fullslice[109:114]))
    hour=bin2dec(reverse(fullslice[115:119]))
    slice_count=bin2dec(reverse(fullslice[120:127]))
```

Appendix E: Revisions to Manual

Rev. Date	Rev No.	Summary	Section
3-30-11	B	Inserted information on pair-wise swapping for GS data	5.2
		Added CIP-GS code samples	Appendices
		Reorganized manual	Throughout

© 2011 DROPLET MEASUREMENT TECHNOLOGIES, INC.