DDF_03 and DDF_04 File Structures and Ethernet Command Structure

Used with DIDSON V5.26.26 Topside Application

(sonar firmware revision 6-14 or higher)

4 June 2018

Bill Hanot
Sound Metrics Corporation
11010 Northup Way
Bellevue, WA 98004
(425) 822-3001

bill@soundmetrics.com

Table of Contents

1 DIDSON Data File Overview for DDF_03 and DDF_04

The DIDSON data file format as indicated by the ***.ddf*** file extension consists of a master file header followed by N frames of data taken at a single frequency. A snapshot file is the special case where N = 1. Each frame of data also starts with its own header.

A file version number is embedded in the master file header so that the DIDSON application can play back older format files while always saving files in the latest format. The file version would increment upon changes to the master file header. A frame version number is embedded in each frame header, which tracks differences in the type of data returned by the sonar. Using two different version numbers allows for user customization of the client application while maintaining uniformity in sonar software.

The sonar firmware may be updated by rewriting the sonar executable file on the CompactFlash card which serves as the C:\ drive for the sonar CPU, using the topside software command *Sonar->Update Firmware*.

The acoustic data in each frame is organized as an array of type char[sample][beam] where sample is always [0..511] and beam is :

DIDSON-Std [LF 0..47, HF 0..95], for a total of (LF) 24576 or (HF) 49152 bytes per frame.
DIDSON-LR [LF 0..47, HF 0..47], for a total of (LF, HF) 24576 bytes per frame.

The following formula may be employed to calculate offsets (in bytes) to the acoustic data in a ***.ddf*** file:

sample_offset = master_header_length + frame_index *frame_length + frame_header_length + sample * NUM_(LF,HF)_BEAMS + beam

1.1 Header Lengths VERSION_DDF_03

fixed_master_header	=	512 bytes
fixed_frame0_header	=	256 bytes
acoustic_data	=	24576 bytes (LF) or 49152 bytes (HF)
fixed_frame1_header	=	256 bytes
acoustic_data	=	24576 bytes (LF) or 49152 bytes (HF)
fixed_frameN_header	=	256 bytes
acoustic_data	=	DIDSON-Std 24576 bytes (LF) or 49152 bytes (HF) (DIDSON-LR 24576 bytes (LF, HF))

1.2 Header Lengths VERSION_DDF_04

fixed_master_header	=	1024 bytes
fixed_frame0_header	=	1024 bytes
acoustic_data	=	24576 bytes (LF) or 49152 bytes (HF)
fixed_frame1_header	=	1024 bytes
acoustic_data	=	24576 bytes (LF) or 49152 bytes (HF)
fixed_frameN_header	=	1024 bytes
acoustic_data	=	DIDSON-Std 24576 bytes (LF) or 49152 bytes (HF)
		(DIDSON-LR 24576 bytes (LF, HF))

1.3 Acoustic Data Array Ordering

The acoustic data itself is stored in a BYTE array of dimensions [SAMPLE][BEAM], where SAMPLE = 0...511, and BEAM = 0...47 (LF), 0...95 (HF) for DIDSON-Std, 0…47 (LF, HF) for DIDSON-LR

For DIDSON-Std using HF, if S = sample and B = beam:

Address	Data
0	S0B0
1	S0B1
2	S0B2
.	.
.	.
.	.
95	S0B95
96	S1B0
97	S1B1
98	S1B2
.	.
.	.
.	.
49151	S511B95

2 Data File Format VERSION_DDF_03

```
enum {VERSION_DDF_03 = 0x03464444};      // reverse order so "DDF" shows up when
                                         // opening binary file (DIDSON V4.00 + later)

FILE STRUCTURE

{
        Header          // 512 bytes
        Frame 0
        Frame 1         // DIDSON-Std HF      = 49408 = 256 (header) + 49152 (96*512 data)
        Frame 2         // DIDSON-Std LF      = 24832 = 256 (header) + 24576 (48*512 data)
        .               // DIDSON-LR LF/HF    = 24832 = 256 (header) + 24576 (48*512 data)
        .
        .
        Frame (Header.m_nFrameTotal - 1)
}



File Header     class CHeader : public CObject
{
public:
        UINT            m_nVersion;             // added for V2.0 code to ensure backward compatibility
        UINT            m_nFrameTotal;          // written just before closing file when in continuous log mode
        UINT            m_nFrameRate;
        BOOL            m_bHighResolution;
        UINT            m_nNumRawBeams;
        float           m_fSampleRate;
        UINT            m_nSamplesPerChannel;
        UINT            m_nReceiverGain;
        UINT            m_nWindowStart;         // values in header are command values (in frame header are
                                                // sonar values)
        UINT            m_nWindowLength;
        BOOL            m_bReverse;             // added for DIDSON_DDF_01
        UINT            m_nSN;                  // added for DIDSON_DDF_01
        Cstring         m_strDate;              // stored on disk as char[32]
        CString         m_strHeaderID;          // stored on disk as char[256]
        int             m_iUserID1;             // Four user ID values displayed on topside
        int             m_iUserID2;             // These values are inserted by user external to
        int             m_iUserID3;             // the sonar.
        int             m_iUserID4;
        UINT            m_nStartFrame;          // for snippet file, from source file
        UINT            m_nEndFrame;            // for snippet file, from source file
        BOOL            m_bTimeLapse;
        UINT            m_nRecordInterval;
        int             m_iRadioSeconds;
        UINT            m_nFrameInterval;
        UINT            m_nFlags;               // save display processing flags (see Note 7)
        UINT            m_nAuxFlags;            // identify sources of aux information present in frame headers
        UINT            m_nSspd;                // sound velocity in water when file is recorded
```

```
UINT        m_n3DFlags;             // identify available sources of orientation data
UINT        m_nSoftwareVersion;     // record software version that created a file
UINT        m_nWaterTemp;           // record temperature selection [0,1,2] for TL on playback
UINT        m_nSalinity;            // record salinity selection [0,1,2] for TL on playback
UINT        m_nPulseLength;         // added for ARIS but not used in DIDSON software
UINT        m_nTxMode;              // added for ARIS but not used in DIDSON software
UINT        m_nVersionFPGA;         // added for ARIS but not used in DIDSON software
UINT        m_nVersionPSuC;         // added for ARIS but not used in DIDSON software
UINT        m_nThumbStartFrame;     // added for ARIS but not used in DIDSON software
UINT        m_nThumbEndFrame;       // added for ARIS but not used in DIDSON software
UINT        m_nExtensionType;       // added for ARIS but not used in DIDSON software
UINT        m_nExtensionLength;     // added for ARIS but not used in DIDSON software
BYTE        m_cRsvdData[MASTER_PAD_LENGTH];    // (588) pad master to 512 bytes
```

Note: the DIDSON sonar has a choice between *Classic Windows* (**CW**) or *Extended Windows* (**XW**), set through the *Edit->Sonar->DidsonAppV5.ini File* command in the topside software. The two modes use different master clock frequencies for data acquisition, leading to differences in *Window Start* increments and *Window Length* values as noted below. The values given in meters are dependent on the sound velocity used, which defaults to 1465 m/s for **CW** and 1457 m/s for **XW**. All Sound Metrics DIDSON sonars default to **XW**.

```
Frame           class CFrame : public CObject
{
public:
        UINT        m_nFrameNumber;
        CTime       m_tFrameTime;       // client PC time (8 bytes, was 4 bytes prior to V4.00)
        UINT        m_nVersion;         // keep unique version for frame data (sonar data format)
                                        // At this time not used. Any change in frame header
                                        // generates an increment in file version
        UINT        m_nStatus;          // feedback on client commands
        UINT        m_nYear;            // sonar CPU year  ex: 2002
        UINT        m_nMonth;           // 1-12
        UINT        m_nDay;             // 1-31
        UINT        m_nHour;            // 0-23
        UINT        m_nMinute;          // 0-59
        UINT        m_nSecond;          // 0-59
        UINT        m_nHSecond;         // 0-99
        UINT        m_nTransmitMode;    // bit1 = ENABLE, bit0 = HF_MODE
        UINT        m_nWindowStart;     // 1-31  CW meters = N*Incr, Incr = .375 (HF) or 0.75 (LF)
                                        //       XW meters = N*Incr, Incr = .42 (HF) or 0.84 (LF)
        UINT        m_nWindowLength;    // 0-3   CW (1.125, 2.25, 4.5, 9 m) HF or (4.5, 9, 18, 36 m) LF
                                        // Std   XW (1.25, 2.5, 5, 10 m) HF or (5, 10, 20, 40 m) LF
                                        // LR    XW (2.5, 5, 10, 20 m) HF or (10, 20, 40, 80 m) LF
        UINT        m_nThreshold;       // 0-80 dB for display only – does not affect raw data
        UINT        m_nIntensity;       // 10-90 dB for display only – does not affect raw data
        UINT        m_nReceiverGain;    // 0-40 dB normalized gain range
        UINT        m_nDegC;            // Power Supply Temp °C
        UINT        m_nDegC2;           // A/D Temp °C
        UINT        m_nHumidity;        // relative humidity 0-100
        UINT        m_nFocus;           // 0-255  Focus position code (range varies with sonar model)
        UINT        m_nBattery;         // Volts at sonar PS in tenths, e.g. 145 = 14.5 volts (Note 12)
        float       m_fUserValue1;      // User defined value for embedded applications (Note 13)
        float       m_fUserValue2;      // User defined value for embedded applications
        float       m_fUserValue3;      // User defined value for embedded applications
        float       m_fUserValue4;      // User defined value for embedded applications
        float       m_fUserValue5;      // User defined value for embedded applications
```

```
    float       m_fUserValue6;      // User defined value for embedded applications
    float       m_fUserValue7;      // User defined value for embedded applications
    float       m_fUserValue8;      // User defined value for embedded applications
    float       m_fVelocity;        // These are space holders for platform generated data
    float       m_fDepth;           // Placed in the header external to the sonar (except DH model)
    float       m_fAltitude;        // Placed in the header external to the sonar
    float       m_fPitch;           // Placed in the header external to the sonar
    float       m_fPitchRate;       // Placed in the header external to the sonar
    float       m_fRoll;            // Placed in the header external to the sonar
    float       m_fRollRate;        // Placed in the header external to the sonar
    float       m_fHeading;         // From sonar compass or external to the sonar
    float       m_fHeadingRate;     // Placed in the header external to the sonar
    float       m_fSonarPan;        // From sonar compass when installed, else X2/PT-25/Sidus
                                    // Renamed to m_fCompassHeading in DDF_04
    float       m_fSonarTilt;       // From sonar compass when installed, else X2/PT-25/Sidus
                                    // Renamed to m_fCompassPitch in DDF_04
    float       m_fSonarRoll;       // From sonar compass when installed, else X2/PT-25/Sidus
                                    // Renamed to m_fCompassRoll in DDF_04
    double      m_dLatitude;        // In degrees ddd.ddddddd
    double      m_dLongitude;       // In degrees ddd.ddddddd
    float       m_fSonarPosition;   // Placed in the header external to the sonar
    UINT        m_nConfigFlags      // sonar configuration flags updated Dec 2006 (see Note 8)
    UINT        m_nPrismTilt        // sonar split-body variable added 26 April 2004
    float       m_fTargetRange;     // added 14 September 2004 for topside software use
    float       m_fTargetBearing;   // added 14 September 2004 for topside software use
    BOOL        m_bTargetPresent;   // added 14 September 2004 for topside software use
    UINT        m_nFirmwareRevision; // added 1 April 2005
    UINT        m_nFlags;           // added 1 August 2005 (see Note 9)
    UINT        m_nSourceFrame;     // added 16 Nov 2006  (record CSOT source frame)
    float       m_fWaterTemp;       // added 19 February 2007 to store NMEA MTW input
    float       m_fSonarX;          // added 26 December 2007
    float       m_fSonarY;          // added 26 December 2007
    float       m_fSonarZ;          // added 6 March 2008
    float       m_fSonarPan;        // added 6 March 2008
    float       m_fSonarTilt;       // added 6 March 2008
    float       m_fSonarRoll;       // added 6 March 2008 (256 bytes, end of V03 frame header)
    BYTE        m_cData[DATA_SIZE]; // acoustic data length
                                    // DIDSON-Std  24576 (LF) or 49152 (HF)
                                    // DIDSON-LR   24576 (LF, HF)
};
```

3 Data File Format VERSION_DDF_04

```
enum {VERSION_DDF_04 = 0x04464444};      // reverse order so "DDF" shows up when
                                         // opening binary file (DIDSON V5.17 + later)

FILE STRUCTURE

{
        Header          // 1024 bytes
        Frame 0
        Frame 1         // DIDSON-Std HF      = 50176 = 1024 (header) + 49152 (96*512 data)
        Frame 2         // DIDSON-Std LF      = 25600 = 1024 (header) + 24576 (48*512 data)
        .               // DIDSON-LR LF/HF    = 25600 = 1024 (header) + 24576 (48*512 data)
        .
        .
        Frame (Header.m_nFrameTotal - 1)
}



File Header     class CHeader : public CObject
{
public:
        UINT            m_nVersion;             // added for V2.0 code to ensure backward compatibility
        UINT            m_nFrameTotal;          // written just before closing file when in continuous log mode
        UINT            m_nFrameRate;
        BOOL            m_bHighResolution;
        UINT            m_nNumRawBeams;
        float           m_fSampleRate;
        UINT            m_nSamplesPerChannel;
        UINT            m_nReceiverGain;
        UINT            m_nWindowStart;         // values in header are command values (in frame header are
                                                // sonar values)
        UINT            m_nWindowLength;
        BOOL            m_bReverse;             // added for DIDSON_DDF_01
        UINT            m_nSN;                  // added for DIDSON_DDF_01
        Cstring         m_strDate;              // stored on disk as char[32]
        CString         m_strHeaderID;          // stored on disk as char[256]
        int             m_iUserID1;             // Four user ID values displayed on topside
        int             m_iUserID2;             // These values are inserted by user external to
        int             m_iUserID3;             // the sonar.
        int             m_iUserID4;
        UINT            m_nStartFrame;          // for snippet file, from source file
        UINT            m_nEndFrame;            // for snippet file, from source file
        BOOL            m_bTimeLapse;
        UINT            m_nRecordInterval;
        int             m_iRadioSeconds;
        UINT            m_nFrameInterval;
        UINT            m_nFlags;               // save display processing flags (see Note 7)
        UINT            m_nAuxFlags;            // identify aux data present in frame headers (see Note 14)
        UINT            m_nSspd;                // sound velocity in water when file is recorded
```

```
UINT        m_n3DFlags;             // flags for 3D processing options (obsolete)
UINT        m_nSoftwareVersion;     // record software version that created a file
UINT        m_nWaterTemp;           // record temperature selection [0,1,2] for TL on playback
UINT        m_nSalinity;            // record salinity selection [0,1,2] for TL on playback
UINT        m_nPulseLength;         // added for ARIS but not used in DIDSON software
UINT        m_nTxMode;              // added for ARIS but not used in DIDSON software
UINT        m_nVersionFPGA;         // added for ARIS but not used in DIDSON software
UINT        m_nVersionPSuC;         // added for ARIS but not used in DIDSON software
UINT        m_nThumbStartFrame;     // added for ARIS but not used in DIDSON software
UINT        m_nThumbEndFrame;       // added for ARIS but not used in DIDSON software
UINT        m_nExtensionType;       // added for ARIS but not used in DIDSON software
UINT        m_nExtensionLength;     // added for ARIS but not used in DIDSON software
BYTE        m_cRsvdData[MASTER_PAD_LENGTH];    // (588) pad master to 512 bytes
```

Note: the DIDSON sonar has a choice between *Classic Windows* (**CW**) or *Extended Windows* (**XW**), set through the *Edit->Sonar->DidsonAppV5.ini File* command in the topside software. The two modes use different master clock frequencies for data acquisition, leading to differences in *Window Start* increments and *Window Length* values as noted below. The values given in meters are dependent on the sound velocity used, which defaults to 1465 m/s for **CW** and 1457 m/s for **XW**. All Sound Metrics DIDSON sonars default to **XW**.

```
Frame Header        class CFrame : public CObject
{
public:
        UINT        m_nFrameNumber;
        CTime       m_tFrameTime;       // client PC time (8 bytes, was 4 bytes prior to V4.00)
        UINT        m_nVersion;         // keep unique version for frame data (sonar data format)
                                        // At this time not used. Any change in frame header
                                        // generates an increment in file version
        UINT        m_nStatus;          // feedback on client commands
        UINT        m_nYear;            // sonar CPU year  ex: 2002
        UINT        m_nMonth;           // 1-12
        UINT        m_nDay;             // 1-31
        UINT        m_nHour;            // 0-23
        UINT        m_nMinute;          // 0-59
        UINT        m_nSecond;          // 0-59
        UINT        m_nHSecond;         // 0-99
        UINT        m_nTransmitMode;    // bit1 = ENABLE, bit0 = HF_MODE
        UINT        m_nWindowStart;     // 1-31   CW meters = N*Incr, Incr = .375 (HF) or 0.75 (LF)
                                        //        XW meters = N*Incr, Incr = .42 (HF) or 0.84 (LF)
        UINT        m_nWindowLength;    // 0-3    CW (1.125, 2.25, 4.5, 9 m) HF or (4.5, 9, 18, 36 m) LF
                                        // Std    XW (1.25, 2.5, 5, 10 m) HF or (5, 10, 20, 40 m) LF
                                        // LR     XW (2.5, 5, 10, 20 m) HF or (10, 20, 40, 80 m) LF
        UINT        m_nThreshold;       // 0-80 dB for display only – does not affect raw data
        UINT        m_nIntensity;       // 10-90 dB for display only – does not affect raw data
        UINT        m_nReceiverGain;    // 0-40 dB normalized gain range
        UINT        m_nDegC;            // Power Supply Temp °C
        UINT        m_nDegC2;           // A/D Temp °C
        UINT        m_nHumidity;        // relative humidity 0-100
        UINT        m_nFocus;           // 0-255  Focus position code (range varies with sonar model)
        UINT        m_nBattery;         // Volts at sonar PS in tenths, e.g. 145 = 14.5 volts (Note 12)
        float       m_fUserValue1;      // User defined value for embedded applications (Note 13)
        float       m_fUserValue2;      // User defined value for embedded applications
        float       m_fUserValue3;      // User defined value for embedded applications
        float       m_fUserValue4;      // User defined value for embedded applications
        float       m_fUserValue5;      // User defined value for embedded applications
        float       m_fUserValue6;      // User defined value for embedded applications
```

```
float       m_fUserValue7;      // User defined value for embedded applications
float       m_fUserValue8;      // User defined value for embedded applications
float       m_fVelocity;        // These are space holders for platform generated data
float       m_fDepth;           // Placed in the header external to the sonar (except DH model)
float       m_fAltitude;        // Placed in the header external to the sonar
float       m_fPitch;           // Placed in the header external to the sonar
float       m_fPitchRate;       // Placed in the header external to the sonar
float       m_fRoll;            // Placed in the header external to the sonar
float       m_fRollRate;        // Placed in the header external to the sonar
float       m_fHeading;         // From sonar compass or external to the sonar
float       m_fHeadingRate;     // Placed in the header external to the sonar
float       m_fCompassHeading;  // Was m_fSonarPan in DDF_03
float       m_fCompassPitch;    // Was m_fSonarTilt in DDF_03
float       m_fCompassRoll;     // Was m_fSonarRoll in DDF_03
double      m_dLatitude;        // In degrees ddd.ddddddd
double      m_dLongitude;       // In degrees ddd.ddddddd
float       m_fSonarPosition;   // Placed in the header external to the sonar
UINT        m_nConfigFlags      // sonar configuration flags updated Dec 2006 (see Note 8)
UINT        m_nPrismTilt        // sonar split-body variable added 26 April 2004
float       m_fTargetRange;     // added 14 September 2004 for topside software use
float       m_fTargetBearing;   // added 14 September 2004 for topside software use
BOOL        m_bTargetPresent;   // added 14 September 2004 for topside software use
UINT        m_nFirmwareRevision; // added 1 April 2005
UINT        m_nFlags;           // added 1 August 2005 (see Note 9)
UINT        m_nSourceFrame;     // added 16 Nov 2006  (record CSOT source frame)
float       m_fWaterTemp;       // added 19 February 2007 to store NMEA MTW input
UINT        m_nTimerPeriod;     // added 18 June 2007 for non-integral frame rates
float       m_fSonarX;          // added 26 December 2007
float       m_fSonarY;          // added 26 December 2007
float       m_fSonarZ;          // added 6 March 2008 for DDF_04
float       m_fSonarPan;        // from ROS/Sidus/SMC rotators
float       m_fSonarTilt;       // from ROS/Sidus/SMC rotators
float       m_fSonarRoll;       // from ROS/Sidus/SMC rotators
float       m_fPanPNNL;         // legacy custom data from Battelle PNNL equipment
float       m_fTiltPNNL;        // legacy custom data from Battelle PNNL equipment
float       m_fRollPNNL;        // legacy custom data from Battelle PNNL equipment
double      m_dVehicleTime;     // Bluefin HAUV time (seconds since 1 Jan 1970)
float       m_fTimeGGK;         // time data from NMEA GGK string
UINT        m_nDateGGK;         // time data from NMEA GGK string
UINT        m_nQualityGGK;      // date data from NMEA GGK string
UINT        m_nNumSatsGGK;      // number of satellites data from NMEA GGK string
float       m_fDOPGGK;          // degree of precision data from NMEA GGK string
float       m_fEHTGGK;          // data from NMEA GGK string
float       m_fHeaveTSS;        // heave data from TSS aux input string
UINT        m_nYearGPS;         // GPS input data - GPS time data
UINT        m_nMonthGPS;        // GPS input data - GPS time data
UINT        m_nDayGPS;          // GPS input data - GPS time data
UINT        m_nHourGPS;         // GPS input data - GPS time data
UINT        m_nMinuteGPS;       // GPS input data - GPS time data
UINT        m_nSecondGPS;       // GPS input data - GPS time data
UINT        m_nHSecondGPS;      // GPS input data - GPS time data
float       m_fSonarPanOffset;  // sonar mount "pan 0" rotation about Z axis
float       m_fSonarTiltOffset; // sonar mount "tilt 0" rotation about local sonar Y axis
float       m_fSonarRollOffset; // sonar mount "roll 0" rotation about local sonar X axis
float       m_fSonarXOffset;    // added 28 May 2008
float       m_fSonarYOffset;    // added 28 May 2008
```

```
        float       m_fSonarZOffset;        // added 28 May 2008
        float       m_fTMatrix[16];         // added 28 May 2008
        BYTE        m_cRsvdData[FRAME_PAD_LENGTH];// (604) pad frame header to 1024 bytes
        BYTE        m_cData[DATA_SIZE];  // acoustic data length
                                         // DIDSON-Std  24576 (LF) or 49152 (HF)
                                         // DIDSON-LR   24576 (LF, HF)
};
```

4 Metacode Examples

The following code fragments are meant to illustrate the VERSION_DDF_03 and VERSION_DDF_04 disk file formats only and have been simplified in places to remove extraneous detail relating to other formats.

```
void CDidsonDoc::OpenLogFile(const char *logFileName)
{
      m_Header.m_nSoftwareVersion = SOFTWARE_VERSION;

      m_Header.m_nFlags = VALID_FLAGS_ID;

      if (m_bSaveDisplayedDataOnly)
          m_Header.m_nFlags |= SAVE_DISPLAYED;
      else
          m_Header.m_nFlags &= ~SAVE_DISPLAYED;

      if (m_bCorrectTL)
          m_Header.m_nFlags |= HDR_CORRECT_TL;
      else
          m_Header.m_nFlags &= ~HDR_CORRECT_TL;

      if (pView->m_bSubtractBackground)
          m_Header.m_nFlags |= SUBTRACT_BACK;
      else
          m_Header.m_nFlags &= ~SUBTRACT_BACK;

      if (m_bDetectMotion)
          m_Header.m_nFlags |= DETECT_MOTION;
      else
          m_Header.m_nFlags &= ~DETECT_MOTION;

      if (m_bLongRange)
          m_Header.m_nFlags |= HDR_LONG_RANGE;
      else
          m_Header.m_nFlags &= ~HDR_LONG_RANGE;

      if (m_b3000m)
          m_Header.m_nFlags |= HDR_3000M;
      else
          m_Header.m_nFlags &= ~HDR_3000M;

      if (m_bSplitBody)
          m_Header.m_nFlags |= HDR_SPLIT_BODY;
      else
          m_Header.m_nFlags &= ~HDR_SPLIT_BODY;
```

```
if (m_bHiResLens)
    m_Header.m_nFlags |= HDR_BIG_LENS;
else
    m_Header.m_nFlags &= ~HDR_BIG_LENS;

if (m_bAuxLens)
    m_Header.m_nFlags |= HDR_AUX_LENS;
else
    m_Header.m_nFlags &= ~HDR_AUX_LENS;

if (m_bCompass)
    m_Header.m_nFlags |= COMPASS_INSTALLED;
else
    m_Header.m_nFlags &= ~COMPASS_INSTALLED;

if (m_bMagneticHeading)
    m_Header.m_nFlags |= MAGNETIC_HEADING;
else
    m_Header.m_nFlags &= ~MAGNETIC_HEADING;

if (m_bRecordVoiceAnnotation)
    m_Header.m_nFlags |= AUDIO_NOTES;
else
    m_Header.m_nFlags &= ~AUDIO_NOTES;

m_logFile.Write(&m_Header.m_nVersion, sizeof(UINT));
m_logFile.Write(&m_Header.m_nFrameTotal, sizeof(UINT));
m_logFile.Write(&m_Header.m_nFrameRate, sizeof(UINT));
m_logFile.Write(&m_Header.m_bHighResolution, sizeof(BOOL));
m_logFile.Write(&m_Header.m_nNumRawBeams, sizeof(UINT));
m_logFile.Write(&m_Header.m_fSampleRate, sizeof(float));
m_logFile.Write(&m_Header.m_nSamplesPerChannel, sizeof(UINT));
m_logFile.Write(&m_Header.m_nReceiverGain, sizeof(UINT));
m_logFile.Write(&m_Header.m_nWindowStart, sizeof(UINT));
m_logFile.Write(&m_Header.m_nWindowLength, sizeof(UINT));
p = m_Header.m_strDate.GetBufferSetLength(STR_DATE_LENGTH);
m_logFile.Write(p, STR_DATE_LENGTH);
p = m_Header.m_strHeaderID.GetBufferSetLength(STR_ID_LENGTH);
m_logFile.Write(p, STR_ID_LENGTH);
m_logFile.Write(&m_Header.m_iUserID1, sizeof(UINT));
m_logFile.Write(&m_Header.m_iUserID2, sizeof(UINT));
m_logFile.Write(&m_Header.m_iUserID3, sizeof(UINT));
m_logFile.Write(&m_Header.m_iUserID4, sizeof(UINT));
m_logFile.Write(&m_Header.m_nStartFrame, sizeof(UINT));
m_logFile.Write(&m_Header.m_nEndFrame, sizeof(UINT));
m_logFile.Write(&m_Header.m_bTimeLapse, sizeof(BOOL));
m_logFile.Write(&m_Header.m_nRecordInterval, sizeof(UINT));
m_logFile.Write(&m_Header.m_iRadioSeconds, sizeof(int));
m_logFile.Write(&m_Header.m_nFrameInterval, sizeof(UINT));
m_logFile.Write(&m_Header.m_nFlags, sizeof(UINT));
m_logFile.Write(&m_Header.m_ nAuxFlags, sizeof(UINT));
m_logFile.Write(&m_Header.m_ nSspd, sizeof(UINT));
m_logFile.Write(&m_Header.m_n3DFlags, sizeof(UINT));
m_logFile.Write(&m_Header.m_nSoftwareVersion, sizeof(UINT));
m_logFile.Write(&m_Header.m_nWaterTemp, sizeof(UINT));
m_logFile.Write(&m_Header.m_nSalinity, sizeof(UINT));
m_logFile.Write(&m_Header.m_nPulseLength, sizeof(UINT));
```

```
      m_logFile.Write(&m_Header.m_nTxMode, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nVersionFPGA, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nVersionPSuC, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nThumbStartFrame, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nThumbEndFrame, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nExtensionType, sizeof(UINT));
      m_logFile.Write(&m_Header.m_nExtensionLength, sizeof(UINT));
      if (m_Header.m_nVersion < VERSION_DDF_04)
            m_logFile.Write(&m_Header.m_cRsvdData[0], MASTER_PAD_LENGTH-512);
      else
            m_logFile.Write(&m_Header.m_cRsvdData[0], MASTER_PAD_LENGTH);

 }

void CDidsonDoc::AppendFrame()      // called for logging data directly to disk
{
    // write frame header followed by frame data (reduced to simplest case)

    m_nDataSize = m_Header.m_nNumRawBeams * m_Header.m_nSamplesPerChannel;

    if (m_Header.m_nVersion == VERSION_DDF_03)
    {
      m_Frame.m_nVersion = VERSION_DDF_03;
      m_logFile.Write((char *)&m_Frame.m_nFrameNumber, FRM03_HDR_SIZE); // 256
      m_logFile.Write((char *)m_Frame.m_cData, m_nDataSize);

    }
    else if (m_Header.m_nVersion == VERSION_DDF_04)
    {
      m_Frame.m_nVersion = VERSION_DDF_04;
      m_logFile.Write((char *)&m_Frame.m_nFrameNumber, FRM04_HDR_SIZE); // 1024
      m_logFile.Write((char *)m_Frame.m_cData, m_nDataSize);

    }
}
```

...when it's time to close the file, write the actual number of frames to master header

```
nTotal = (m_Header.m_nFrameTotal == 0) ? nCurrent : m_Header.m_nFrameTotal;
m_logFile.Seek(4L, CFile::begin);         // point to m_Header.m_nFrameTotal
m_logFile.Write(&nTotal, sizeof(UINT));   // update frame total in header
m_logFile.Flush();
m_logFile.Close();
```

5 DIDSON V5.26 Ethernet Command Structure

Revised Edition
4 June 2018

(documents sonar firmware revision 6-14 or higher)

5.1 Client → Server Communication

DIDSON Ethernet commands and data are sent to the sonar in the format:

WORD	nCommand	the command type to be executed in the sonar
WORD	nSize	the number of bytes following nCommand, nSize, nPktType, nPktNum
WORD	nPktType	code for data to be returned (see below)
WORD	nPktNum	number of packet to be returned

The list of command values…

Value	Alias	Description	Associated Size
0xa501	C_CLOSE_CONNECTION	Client (PC) closing connection	0
0xa502	C_DATA	Request data from sonar	0
0xa504	C_FILE	Reserved	0
0xa508	C_NETINFO	Query sonar network parameters	0
0xa510	C_COMMAND	Change sonar parameter(s)	2 * num_cmd_pairs
0xa520	C_LOAD_PARAMS	Download sonar configuration	sizeof(PARAMS)

Typically the C_DATA command is sent at the desired frame rate, and may be ORed with C_COMMAND or C_LOAD_PARAMS when additional actions are requested. In other words, one may request a new frame and change gain with the same command transmission. The C_FILE command is used for managing locally recorded .ddf files.

The list of packet type values…

Value	Alias	Description	Associated Size
0x0001	HF1_REQUEST	Frame header + HF_DATA/2	24832
0x0002	HF2_REQUEST	HF_DATA/2	24576
0x0004	LF_REQUEST	Frame header + LF_DATA	24832
0x0008	HDR_REQUEST	Frame header	256
0x0010	PKT_REQUEST	DATA Packet	1024
0x0020	NEW_PING	Start new ping cycle	0
0x0040	UPDATE_FRAME_HEADER	Stuff frame header with status data	sizeof(UPDATE)
0x0400	BURST_REQUEST	Get complete frame via packets	[1…100] see below
0x0001	FILE_LIST	List of local .ddf file sizes	N*4
0x0002	FILE_REQUEST	Download file N	0
0x0004	FILE_ERASE	Erase file N (N == 9999 all files)	0
0x0008	NEXT_PACKET	Send next file packet	1024

There are three modes of data transmission supported: frame transfer and packet transfer (normal or burst).

DIDSON-Std
In HF frame transfer mode a C_DATA command with packet type HF1_REQUEST is made, followed by another C_DATA command with packet type (HF2_REQUEST | NEW_PING) after reception of the first half of the frame data. Note that only one NEW_PING command is used per pair of data transfer requests. In LF mode the packet type would be (LF_REQUEST | NEW_PING) for every transfer.

DIDSON-LR
In HF frame transfer mode a C_DATA command with packet type (HF1_REQUEST | NEW_PING) is made for each frame. In LF mode the packet type would be (LF_REQUEST | NEW_PING) for every transfer.

Frame transfer is the fastest mode, allowing HF frame rates of up to 24/s using 100BaseT and 13/s using 10BaseT (or 10Base2) transmission protocols (assuming the shortest data collection window and sonar video data display disabled).

The normal packet transfer mode involves 25 (LF) or 49 (HF, DIDSON-Std) request/data cycles to transfer one frame of data. In this case the nPktNum parameter is set for each data packet, and if the corresponding data is not returned, the packet is re-requested. For example, a DIDSON-Std HF frame is retrieved with the sequence (values for nCmd, nSize, nPktType, nPktNum):

C_DATA, 0, HDR_REQUEST, 0
C_DATA, 0, PKT_REQUEST, 0
C_DATA, 0, PKT_REQUEST, 1
.
.
.
C_DATA, 0, PKT_REQUEST, 46
C_DATA, 0, PKT_REQUEST | NEW_PING, 47

Normal packet transfer is a more robust mode, as dropped packets may be re-requested without starting a new frame cycle or retransferring an entire frame. The frame rates are limited (once again assuming the shortest data collection window and sonar video data display disabled) to 15/s for 100BaseT transmission and 9/s for 10BaseT (or 10Base2) transmission.

Burst mode packet transfer allows the (packet) retrieval of an entire image frame with a single data request. First the 256-byte header is returned, followed by 24 (Std-LF, LR-LF/HF) or 48 (Std-HF) 1024-byte data packets. The *nPktNum* parameter is used to send an inter-packet delay factor (valid 1…100, default is 10) that controls the overall time to return the image frame. The C_DATA command is sent to the sonar with *nPktType* set to (BURST_REQUEST | NEW_PING) and *nPktNum* set as just mentioned.

Normally, C_DATA commands are sent with the *nSize* parameter set to zero, and the requested data blocks are selected with the *nPktType* variable (with *NEW_PING* ORed with the data request to start the next frame acquisition). For embedded control cases where the user wishes to combine other platform status data with the returned DIDSON image frames (such as velocity, depth, altitude, pitch, roll, etc.) there is the UPDATE_FRAME_HEADER flag which may also be ORed with *nPktType*. In this case the *nSize* parameter is set to *sizeof(struct UPDATE_HEADER)*, and the *UPDATE_HEADER* structure is filled in with the requisite information before appending to the command array. The appropriate *m_nUpdateFlags* bit(s) are set to reflect which parameters are transferred to the sonar frame header before the image frame is returned.

The following code segment (simplified) illustrates building and sending a standard image frame request, with or without the additional *UPDATE_HEADER* structure data:

```
if (m_bEthernetMode)                // standard frame request
{
      cmdTime = timeGetTime();
      m_nClientCmd = C_DATA;
```

```
if (m_bTransferByPacket)
{
      if (m_bCommandOnly)     // always NEW_PING but no returned data
            m_nClientPktType = NEW_PING;
      else
            m_nClientPktType = HDR_REQUEST;
      if (m_bFillFrameHeader || m_bClearFrameHeader)
            m_nClientPktType |= UPDATE_FRAME_HEADER;
}
else
{
      if (m_bCommandOnly)     // always NEW_PING but no returned data
      {
            m_nClientPktType = NEW_PING;
      }
      else
      {
            if (m_bLongRange)
            {
                  if (m_Header.m_bHighResolution)
                        m_nClientPktType = HF1_REQUEST | NEW_PING;
                  else
                        m_nClientPktType = LF_REQUEST | NEW_PING;
            }
            else
            {
                  if (m_Header.m_bHighResolution)
                        m_nClientPktType = HF1_REQUEST;
                  else
                        m_nClientPktType = LF_REQUEST | NEW_PING;
            }
            if (m_bFillFrameHeader || m_bClearFrameHeader)
                  m_nClientPktType |= UPDATE_FRAME_HEADER;
      }
}
m_nClientPktNum = 0;
m_nClientSize = m_bFillFrameHeader || m_bClearFrameHeader
                ? sizeof(struct CDidsonDoc::UPDATE_HEADER) : 0;
memcpy(m_ClientData+0, &m_nClientCmd, sizeof(WORD));
memcpy(m_ClientData+2, &m_nClientSize, sizeof(WORD));
memcpy(m_ClientData+4, &m_nClientPktType, sizeof(WORD));
memcpy(m_ClientData+6, &m_nClientPktNum, sizeof(WORD));
if (m_bFillFrameHeader) // load test data
{
      m_upd.m_nUpdateFlags = UPDATE_VELOCITY | UPDATE_DEPTH |
                  UPDATE_ALTITUDE | UPDATE_LATITUDE | UPDATE_LONGITUDE;
      m_upd.m_fVelocity = simulated data;
      m_upd.m_fDepth = simulated data;
      m_upd.m_fAltitude = simulated data;
      m_upd.m_dLatitude = simulated data;
      m_upd.m_dLongitude = simulated data;
      memcpy(m_ClientData+8, (BYTE*)&m_upd, sizeof(struct UPDATE_HEADER));
}
else if (m_bClearFrameHeader)
{
      m_bClearFrameHeader = FALSE;
      m_upd.m_nUpdateFlags = UPDATE_VELOCITY | UPDATE_DEPTH |
```

```
                        UPDATE_ALTITUDE | UPDATE_LATITUDE | UPDATE_LONGITUDE;
            m_upd.m_fVelocity = 0.0;
            m_upd.m_fDepth = 0.0;
            m_upd.m_fAltitude = 0.0;
            m_upd.m_dLatitude = 0.0;
            m_upd.m_dLongitude = 0.0;
            memcpy(m_ClientData+8, (BYTE*)&m_upd, sizeof(struct UPDATE_HEADER));
      }
      m_pClientSocket->DoAsyncSendBuff(); // send Ethernet command
}
```

When C_COMMAND is used to indicate that one or more sonar parameters are to be changed, valid command pairs (DWORD Command, DWORD Value) are:

Command	Alias	Description	Valid Values
0x00	WINDOW_START	Set acoustic window start range	1...31
0x01	WINDOW_LENGTH	Set acoustic window length	0...3
0x02	THRESHOLD	Set display threshold	0...80
0x03	INTENSITY	Set display intensity	10...90
0x04	RECEIVER_GAIN	Set receiver gain	0...40
0x05	PALETTE	Set display palette	(Note 11)
0x06	DISPLAY_MODE	Set sonar display mode	(Note 1)
0x07	PLAYBACK_IMAGE	File number for sonar playback	obsolete
0x08	TRANSMIT_MODE	Enable transmit & Set HF/LF	(Note 2)
0x09	SAVE_IMAGE	Save current frame to file	obsolete
0x0a	DOWNLOAD_IMAGES	Download saved files via RS-232	obsolete
0x0b	ERASE_IMAGES	Erase saved files in sonar	obsolete
0x0c	SET_YEAR	Set year in sonar RTC	2000...2038
0x0d	SET_MONTH	Set month in sonar RTC	1...12
0x0e	SET_DAY	Set day in sonar RTC	1...31
0x0f	SET_HOUR	Set hour in sonar RTC	0...23
0x10	SET_MINUTE	Set minute in sonar RTC	0...59
0x11	SET_SECOND	Set second in sonar RTC	0...59
0x12	EXIT_TO_DOS	Quit to sonar command prompt	don't care
0x13	FOCUS	Move sonar lens to new position	5…250
0x14	RETRACT_LENS	Withdraw sonar lens to home position	don’t care
0x15	DISPLAY_REVERSE	Reverse NTSC video image L-R	0...1
0x16	DISPLAY_GRID	Overlay grid on NTSC video	0...1
0x17	DISPLAY_STATUS1	Display status string on NTSC video	(Note 3)
0x18	DISPLAY_STATUS2	Display status string on NTSC video	(Note 3)
0x19	BAUD_RATE	Set baud rate for next sonar power-up	(Note 4)
0x1a	SET_UNITS	Set units (English, Metric, Yards)	0…2
0x1b	TILT	Set prism tilt (Split-body only)	0…255
0x1c	QUERY_STATUS	Request simple Ethernet return	don’t care
0x1d	FORMAT_HD	Format internal sonar hard drive	don’t care
0x1e	COMPENSATE_COMPASS	Perform compass compensation	don’t care
0x1f	SET_DECLINATION	Set compass declination value	(Note 10)
0x20	COMPASS_STATUS	Request compass status return	don’t care
0x21	ZERO_COMPASS_MOUNT	Remove tilt/roll offsets when level	don’t care
0x22	PTR_MOTOR	X2 rotator motor command	(Note 16)
0x23	PTR_SPEED	X2 rotator velocity command	
0x24	OMAP_PULSE_LENGTH	Obsolete was test only	
0x25	OMAP_SAMPLE_RATE	Obsolete was test only	
0x26	OMAP_TX_MODE	Obsolete was test only	

The topside application sends command codes (0x00...0x1d) ORed with the COMMAND_PATTERN code 0x695a0000 before sending to the sonar to validate the command. The sonar code checks for the pattern 0x695a in the high word before accepting a command. Normally the entire command array is sent for every command transmission (e.g. 0x22 = 34 command, value pairs…272 bytes) with only the desired commands set to their array position to indicate an active command.

Use the C_FILE command to download or erase local HD .ddf files recorded during autonomous mode. The packet code FILE_LIST will return a list of N file sizes (return packet size is N*4), where N is the total number of .ddf files found on logical drives D:\ through G:\. Subsequent C_FILE commands combined with packet codes FILE_REQUEST or FILE_ERASE will download or erase file N where N is sent as the nPktNum parameter. For downloading the client requests additional packets (ORing NEXT_PACKET with FILE_REQUEST) until (file size) bytes have been returned. The packets returned are 1024 bytes until the final packet, which may be smaller.

5.2 Server → Client Communication

The sonar returns a 4-word (mininum) packet:

WORD	nCommand	the type of data returned from the sonar
WORD	nSize	8/12 (nCommand, nSize, nPktType, nPktNum/nFilePktNum) + data size
WORD	nPktType	code for data returned
WORD	nPktNum	number of packet returned (for C_DATA commands)
or		
DWORD	nFilePktNum	number of packet returned (for C_FILE commands)

Value	Alias	Description	Associated Size
0x5a01	S_CLOSE_CONNECTION	Server (sonar) closing connection	0
0x5a02	S_DATA	Server (sonar) returning data	varies
0x5a04	S_TIMEOUT	Server (sonar) returning error	0
0x5a08	S_FILE	Server (sonar) returning file data	1024 (max)
0x5a10	S_NETINFO	Server (sonar) returning network data	`8+sizeof(struct PARAMS)`
0x5a20	S_QUERY_STATUS	Server (sonar) responding to query	0
0x5a40	S_COMPASS_STATUS	Server (sonar) compass status	0

The 0x5a in the upper byte of the command code validates the data as a sonar command. The acoustic data returned is in the frame format listed above in the file format specification.

The sonar ***frame.status*** return is normally 0x00000000 unless a command sequence has been initiated by the client PC. The following status codes are detected by the host PC software and briefly displayed over the sonar image to provide feedback that the commands have been accepted and executed.

START_CHANGED	0x00000001
LENGTH_CHANGED	0x00000002
THRESHOLD_CHANGED	0x00000004
INTENSITY_CHANGED	0x00000008
GAIN_CHANGED	0x00000010
PALETTE_CHANGED	0x00000020
MODE_CHANGED	0x00000040
PB_IMAGE_CHANGED	0x00000080

```
XMIT_EN_CHANGED         0x00000100
IMAGE_SAVED             0x00000200
IMAGES_DOWNLOADING      0x00000400
IMAGES_ERASED           0x00000800
DATE_CHANGED            0x00001000
TIME_CHANGED            0x00002000
EXITED_TO_DOS           0x00004000
FOCUS_CHANGED           0x00008000
LENS_RETRACTED          0x00010000
SERVER_CMD_ERROR        0x00020000
STRING1_CHANGED         0x00040000
STRING2_CHANGED         0x00080000
AUX_CMD_TIMEOUT         0x00100000
CMD_VALUE_ERROR         0x00200000
PATTERN_ERROR           0x00400000
CMD_SYNC_ERROR          0x00800000
S_READ_ERROR            0x01000000
SWITCH_HIT              0x02000000

struct PARAMS
{
      unsigned int      nPort;            // normally 700
      unsigned int      nSspd;            // meters/second
      unsigned int      nTimeout;         // seconds to wait for connection
      unsigned int      m_nSonarRecordMode; // control flags, see Note 5
      unsigned int      m_nSonarFramesPerFile;
      unsigned int      m_nSonarFrameRate;
      char              m_cIPaddress[16]; // 128.95.97.227 is default
      char              m_cNetmask[16];   // 255.255.0.0 is default
      char              m_cGateway[16];   // not normally used
      unsigned int      m_nSN;            // sonar serial number
      unsigned int      m_nMclk;          // 0=Ext windows, 1=Std windows
      unsigned int      m_nWaterTemp;     // 0=0-10C, 1=10-20C, 2=20 to 30C
      unsigned int      m_nSalinity;      // 0=Fresh, 1=Brackish, 2=Salt
      unsigned int      m_nFocusMode;     // see Note 6
      unsigned int      m_bLongRange;     // .70/1.2MHz model if TRUE
      unsigned int      m_bPrism;         // for Split-Body only
      unsigned int      m_b3000m;         // for 3000m depth version only
      unsigned int      m_bBroadcast;     // use Broadcast IP address to send data
      unsigned int      m_bEtherdata;     // send Ethernet data in Autonomous mode
      unsigned int      m_nSkipCount;     // send Ethernetdata every Nth frame
      unsigned int      m_bCompass;       // compass installed
      unsigned int      m_bHiResLens;     // "Big Lens" installed
      unsigned int      m_bSplitBody;     // Splitbody model
      unsigned int      m_nConfigFlags;   // added 9/7/2007 (see Note 15)
      unsigned int      m_nMinutesOn;     // added 11/13/2009
      float             m_fPtrLimits[6];  // added 02/02/2010

 } prm;

struct UPDATE_HEADER
{
      float             m_fVelocity;
      float             m_fDepth;
      float             m_fAltitude;
      float             m_fPitch;
```

```
      float             m_fPitchRate;
      float             m_fRoll;
      float             m_fRollRate;
      float             m_fHeading;
      float             m_fHeadingRate;
      float             m_fSonarPan;
      float             m_fSonarTilt;
      float             m_fSonarRoll;
      double            m_dLatitude;
      double            m_dLongitude;
      float             m_fSonarPosition;
      float             m_fTargetRange;
      float             m_fTargetBearing;
      unsigned int      m_bTargetPresent;
      unsigned int      m_nUpdateFlags;   // see flag bit definitions below
      float             m_fUser1;
      float             m_fUser2;
      float             m_fUser3;
      float             m_fUser4;
      float             m_fUser5;
      float             m_fUser6;
      float             m_fUser7;
      float             m_fUser8;
      unsigned int      m_nDegC2;
      unsigned int      m_nFrameNumber;
      float             m_fWaterTemp;
      float             m_fSonarX;
      float             m_fSonarY;
      float             m_fSonarZ;
      double            m_dVehicleTime;
} update;


#define UPDATE_VELOCITY       0x00000001  // UPDATE_HEADER structure flags
#define UPDATE_DEPTH          0x00000002
#define UPDATE_ALTITUDE       0x00000004
#define UPDATE_PITCH          0x00000008
#define UPDATE_PITCHRATE      0x00000010
#define UPDATE_ROLL           0x00000020
#define UPDATE_ROLLRATE       0x00000040
#define UPDATE_HEADING        0x00000080
#define UPDATE_HEADINGRATE    0x00000100
#define UPDATE_SONARPAN       0x00000200
#define UPDATE_SONARTILT      0x00000400
#define UPDATE_SONARROLL      0x00000800
#define UPDATE_LATITUDE       0x00001000
#define UPDATE_LONGITUDE      0x00002000
#define UPDATE_SONARPOSITION  0x00004000
#define UPDATE_TARGETRANGE    0x00008000
#define UPDATE_TARGETBEARING  0x00010000
#define UPDATE_TARGETPRESENT  0x00020000
#define UPDATE_USERDATA       0x00040000
#define UPDATE_SONARTIME      0x00080000
#define UPDATE_DEGC2          0x00100000
#define UPDATE_FRAMENUMBER    0x00200000
#define UPDATE_WATERTEMP      0x00400000
#define UPDATE_SONARX         0X00800000
```

```
#define UPDATE_SONARY          0x01000000
#define UPDATE_SONARZ          0x02000000
#define UPDATE_VEHICLETIME     0x04000000
#define UPDATE_RSVD1           0x08000000
#define UPDATE_RSVD2           0x10000000
#define UPDATE_RSVD3           0x20000000
#define UPDATE_RSVD4           0x40000000
#define UPDATE_RSVD5           0x80000000
```

Notes:

1. Display mode bit codes:

Bit11 = show msgs	bit10 = show horizon	bit9 = svga (1)/vga (0)	bit8 = correct TL
bit7 = smoothing	bit6 = show status	bit5 = auto-resolution	bit4 = HF (1)/LF (0)
bit3 = show range	bit2 = show controls	bit1 = show image	bit0 = realtime (1)/playback (0)

2. Transmit mode bit codes:

Bit1 = tx enable	bit0 = HF (1), LF (0)

3. Used for serial control only (obsolete). Use the C_LOAD_STATUS1 and C_LOAD_STATUS2 commands.

4. Value 0 = 9600 baud, 1 = 38400 baud. Rate is written to sonar didson.ini file and takes effect after the next sonar power-up. (Obsolete)

5. Sonar record mode bit codes: bit1 = REC_EXT_ENABLE, bit0 = DISK_RECORD

6. Focus mode bit codes:

```
bit7-4 = unused                 // reserved for future use
bit3 = UPDATE_CONFIG            // 0=sonar keeps current SN, LR, Prism settings in Config.ini file
                                // 1=update sonar Config.ini file

bit2 = ADVANCED_SCREEN          // set default video output display
bit1 = AUTO_FOCUS               // sonar will auto-focus when set
bit0 = BEAM_AVERAGED            // best focus across FOV when set, best in center when cleared
```

7. Master Header m_nFlags bit codes:

```
VALID_FLAGS_ID      = 0x46000000     // tag for valid m_nFlags entry (0x46 = 'F')
HDR_RESERVED_1      = 0x00000001
HDR_RESERVED_2      = 0x00000002
HDR_CORRECT_TL      = 0x00000004
HDR_LONG_RANGE      = 0x00000008
HDR_PRISM           = 0x00000010
HDR_3000M           = 0x00000020
CSOT_FILE           = 0x00000040
AUDIO_NOTES         = 0x00000080
HDR_SPLIT_BODY      = 0x00000100
HDR_BIG_LENS        = 0x00000200
HDR_AUX_LENS        = 0x00000400
HDR_EXT_WS          = 0x00000800
SAVE_DISPLAYED      = 0x00010000
```

```
DETECT_MOTION      = 0x00020000
SUBTRACT_BACK      = 0x00040000
COMPASS_INSTALLED  = 0x00100000     // flag for compass in master header
MAGNETIC_HEADING   = 0x00200000     // flag for magnetic heading in sonarpan
```

8. Frame Header m_nConfigFlags bit codes:

```
STANDARD_WINDOWS  = 0x00000001 // std = 4.000MHz master clock, ext = 3.580MHz
LONG_RANGE        = 0x00000002 // LR type sonar
AUTO_GAIN_LOW     = 0x00000004 // flag for low (1%) threshold of samples > 200 mu
AUTO_GAIN_MED     = 0x00000008 // flag for med (3%) threshold of samples > 200 mu
AUTO_GAIN_HIGH    = 0x00000010 // flag for high (1%) threshold of samples > 200 mu
ZOOM_2X           = 0x00000020 // zoom factor of 2x in DH internally recorded file
ZOOM_4X           = 0x00000040 // zoom factor of 4x in DH internally recorded file
ZOOM_8X           = 0x00000080 // zoom factor of 8x in DH internally recorded file
SVGA_OUT          = 0x00000200 // sonar video output mode set to SVGA 800x600
DH_MODEL          = 0x00000400 // flag for DH model
MOTOR_NODE1       = 0x00000800 // flag for SMC rotator motor1 present
MOTOR_NODE2       = 0x00001000 // flag for SMC rotator motor2 present
MOTOR_NODE3       = 0x00002000 // flag for SMC rotator motor3 present
REVERSE_VIDEO     = 0x00004000 // flag for sonar video reversed (beam flip)
NSWCPC_FIRMWARE   = 0x00008000 // flag for NSWC-PC restricted firmware installed
SPAWAR_FIRMWARE   = 0x00010000 // flag for SPAWAR firmware installed
DIVER_FIRMWARE    = 0x00020000 // flag for DiverHeld firmware installed
STANDARD_FIRMWARE = 0x00040000 // flag for Standard firmware installed
SLOW_X2_FIRMWARE  = 0x00080000 // flag for Standard firmware with Slow X2
```

9. Frame Header m_nFlags bit codes:

```
INDEX_MARK       = 0x00000001 // index mark present in current frame
AUDIO_MARK       = 0x00000002 // flag for audio file associated with frame
AUDIO_INDEX_MASK = 0x000fff00 // mask for audio file index value insertion
```

10. `SET_DECLINATION` command value

```
m_cmd.iValue[SET_DECLINATION] = 1000 + (int)(m_prm.m_fDeclination * 10.0 + 0.5);
                                        // valid range -90.0 to 90.0
if (m_bMagneticHeading)                 // ensure positive number
      m_cmd.iValue[SET_DECLINATION] |= MAGNETIC_HEADING;
```

11. `PALETTE` command value

`m_cmd.iValue[PALETTE] =` ((StatusColor[1..15] << 8) | ((RangeColor[1..15]) << 4) | Palette[0..9];

12. The voltage monitor value reflects the output of the primary DC/DC converter (nominal 12 Vdc) in standard DIDSONs with the normal 14-32 Vdc input range. For DH models the voltage value is either the battery level or output of the primary DC/DC converter (nominal 15 Vdc) when powered via the topside box and underwater cable. For models with a 12 Vdc regulated or 14-18 Vdc unregulated input, the voltage monitor measures the actual input voltage, downstream of the input reverse protection diode and fuse.

13. The 8 user parameters are not all available when the sonar is controlled from the topside software and certain aux inputs are enabled. For example, if a sonar has a compass installed and the topside software is also controlling a pan/tilt unit, then the pan and tilt values are stored in user7 and user8. Other aux inputs may

also store information in the user parameter slots (e.g. the GGK/RMC/ZDA NMEA messages store satellite time in these locations).

14. Master Header m_nAuxFlags bit codes:

```
VALID_AUX_ID      = 0x50000000      // tag for valid m_nAuxFlags entry
AUX_RMC           = 0x00000001
AUX_HEAD          = 0x00000002
AUX_RATE          = 0x00000004
AUX_PNNL          = 0x00000008
AUX_AVR           = 0x00000010
AUX_GGA           = 0x00000020
AUX_HDT           = 0x00000040
AUX_VTG           = 0x00000080
AUX_ZDA           = 0x00000100
AUX_STX           = 0x00000200
AUX_TSS           = 0x00000400
AUX_VHW           = 0x00000800
AUX_TERRELLA      = 0x00001000
AUX_PANTILT       = 0x00002000
AUX_DPT           = 0x00004000
AUX_GGK           = 0x00008000
AUX_HDG           = 0x00010000
AUX_MTW           = 0x00020000
AUX_XDRA          = 0x00040000
AUX_XDRP          = 0x00080000
AUX_XYZ_POS       = 0x00100000
```

15. prm.m_nConfigFlags bit codes:

```
PS_36V            = 0x00000001      // new power supply with 9-36V input
AUX_LENS          = 0x00000002      // aux lens installed on swing arm
EXTENDED_WS       = 0x00000004      // Altera EPLD programmed for Ext WS [1...63]
FILE_VERSION_04   = 0x00000008      // flag to record/read/return frames as V04
```

16. X2 Rotator:

The SMC X2 pan/tilt/roll motors have 1000 output counts/revolution
The motors are geared down 175/14976 (.011685363) motor shaft revolutions per motor revolution
The output shaft is geared down by an additional 13.5 times (.000865582 pan/tilt/roll shaft revolution per motor revolution)
Overall, output shaft rpm = .000865582 motor shaft rpm, or 1 shaft revolutions = 1155.291429 motor revolutions
This equals 1,155,291.429 counts per shaft revolution (360 degrees), [EPOS steps (x4) 4,621,165.716 steps per pan/tilt/roll shaft revolution]
Or, 3,209.142858 counts [EPOS steps 12,836.57143] per degree of shaft revolution
Thus, 1000 motor rpm = .865582 shaft rpm = 311.60952 shaft degrees/minute = 5.193492 shaft degrees/second
Rotation Speed (deg/s) = .005193492 degrees/second/rpm
Motor Speed (rpm) = 192.5486744 * Rotation Speed (deg/s)

#define RPM_PER_DEG_PER_SEC 192.5486744 motor rpm per degree/second velocity requested
#define COUNTS_PER_DEG 3209.142858 motor tach counts per degree of shaft rotation

Default motor nodes: Tilt = 1, Roll = 2, Pan = 3

SMC Pan/Tilt/Roll motor command structure
The PTR_MOTOR command value (32-bits) stores the command type in the upper 17 bits,
and the command value in the lower 15-bits
The command value is sign-extended to 16-bits after extracting the command type flag in bit15

#define SMC_STOP_PAN	0x80000000	bit31 - stop pan motor
#define SMC_STOP_TILT	0x40000000	bit30 - stop tilt motor
#define SMC_STOP_ROLL	0x20000000	bit29 - stop roll motor
#define SMC_RUN_PAN	0x10000000	bit28 - run pan motor
#define SMC_RUN_TILT	0x08000000	bit27 - run tilt motor
#define SMC_RUN_ROLL	0x04000000	bit26 - run roll motor
#define SMC_VELOCITY	0x02000000	bit25 - run in velocity mode (no fixed end-point)
#define SMC_ABS_POSITION	0x01000000	bit24 - run to relative position (degrees, not counts)
#define SMC_SET_HOME	0x00800000	bit23 - set home to current position
#define SMC_RESET	0x00400000	bit22 - reset all nodes
		bit[21..20] - pan motor node ID [1..3]
		bit[19..18] - tilt motor node ID [1..3]
		bit[17..16] - roll motor node ID [1..3]
#define SMC_JOYCMD	0x00008000	bit15 - joystick velocity command (do immediate)
		bit[14...0] - command value field (signed)