DESIGN REPORT navico

GENERAL INFORMATION					
Product		**Doc. Ctrl. No.**			
Comp		**By:**	AC	**Date**	Jan 25th, 2010
Models		**Review:**		**Rev**	- 01 -
Title	SLG File Format				

DISTRIBUTION LIST		
☐ Plant manager	☒ Product Engineering	☐ Quality
☐ Test Engineering	☐ Manufacturing Eng.	☐ Production
☐ Purchasing	☐ Production Control	☐ Document Control

Objective

To communicate the protocol used for recording sonar log files in the .sl2 format. To enable third parties to develop tools to make use of the files.

Table of Contents

A. Overview

The .slg file format was created about 10 years ago and has a limited amount of data for debugging and is restricted to recording of the primary frequency data only. The records must all be fixed length to enable the random access required by the tools. There is a single header structure followed by multiple blocks of column data.

B. File Header

The file header structure is as follows:

```
#include <Core/pshpack1.h>
struct tSlgHeader
{
    short m_Major;
    short m_Minor;
    short m_BytesPerSounding;
    long  m_WaveletSize;
};
#include <Core/poppack.h>
```

The Major version for SLG files will be 1. The m_BytesPerSounding field is used the record size for all subsequent records. This header is followed by the first data packet.

C. Data Columns

Data packets contain a header with the column setup information and some digital information for the time of the sounding followed by the rangecells section. The rangecells section starts with a two byte size field that contains the number of rangecells in the packet. Each rangecell is 1 byte of logarithmic amplitude data. They are spaced out evenly in time from the upper limit (in feet) offset supplied in the header to the lower limit (in feet). The assumed two way speed of sound is 2400 ft/s, so any adjustments to actual water conditions could be made from the recorded data.

The digital information preceding the rangecell data contains only the valid parameters with some flags for determining which fields are included. Some data such as position, speed and track are only included in the packet when the data GPS position is updated so they will be spaced about 1s apart. Packets in between the updates will not contain these fields. The code in the next section describes how to load the digital information out and keep track of where the rangecell section will start. Since all data columns are the same total number of bytes, a buffer can simply be filled with each new data column and passed into a function to parse the digital information. The offset to the rangecell data can then be used if desired to read out the number and actual rangecell data.

D. Reading the Digital Information

Here is a simple function for receiving the data pointer and returning all the digital information contained in the header. The returned offset can be used to read the number of rangecells out followed by the rangecells. Note that the start of the next packet will not be directly following the end of the rangecells, but will be offset by the m_BytesPerSounding from the start of the packet. This code assumes the CPU it is compiled on can correctly handled misaligned reads of both integer and floating point values. All data is stored little-endian and a little-endian architecture is assumed. Values are in feet for depth, upper/lower limits, and altitude. The PositionX and PositionY values are in “Lowrance Mercator Meters” in the WGS84 datum and a conversion to degrees is provided in section E. Speed is in knots and the track (direction of GPS travel over ground) is in radians.

```
#define FISH_MASK                   7

#define UPPER_LIMIT_BIT             3
#define WATER_TEMPERATURE_BIT 4
#define TEMPERATURE_2_BIT           5
#define TEMPERATURE_3_BIT           6
#define WATER_SPEED_BIT             7
#define POSITION_BIT                8
#define INVALID_DEPTH_BIT           9           // Store the bit as invalid when set to
preserve backward compatibility
#define SURFACE_DEPTH_BIT           10
#define TOP_OF_BOTTOM_DEPTH_BIT     11
#define CHART_IS_50KHz              12
#define TIME_BIT                    13
#define EXTENDED_GPS_INFO_BIT 14
//#define EXTENDED_INFORMATION      15

#define INVALID_ELEVATION           -10000

// Returns the offset into lpData where the start of the column data is
long FileReadSubHeader(void *lpData,
                           float &UpperLimit, float &LowerLimit,
                           float &Depth, bool &DepthValid,
                           bool &WaterTemperatureValid, float &WaterTemperature,
                           bool &Temperature2Valid, float &Temperature2,
                           bool &Temperature3Valid, float &Temperature3,
                           bool &WaterSpeedValid, float &WaterSpeed,
                           bool &PositionValid, long &PositionY, long &PositionX,
                           short &NumberOfFish, float FishDepths[MAX_NUMBER_OF_FISH],
                           float &SurfaceDepth, bool &SurfaceValid,
                           float &TopOfBottomDepth, bool &TopOfBottomValid,
                           bool &ColumnIs50kHz, bool &TimeIsValid, long &TimeIndex,
                           bool &SpeedTrackValid, float &Speed, float &Track,
                           bool &AltitudeValid, float &Altitude)
{
      unsigned short Flags;

      assert(FISH_MASK == MAX_NUMBER_OF_FISH);

      unsigned char *pData = (unsigned char *)lpData;
      short Offset = 0;

      Flags = *(unsigned short *)(pData + Offset);
      Offset += sizeof(Flags);
      LowerLimit = *(float *)(pData + Offset);
      Offset += sizeof(LowerLimit);
      Depth = *(float *)(pData + Offset);
      Offset += sizeof(Depth);

      NumberOfFish = Flags & FISH_MASK;
      int UpperLimitValid = (Flags & (1 << UPPER_LIMIT_BIT));

// Turn off the "you're forcing a int to bool" warning...
#pragma warning(disable : 4800)
      WaterTemperatureValid = (Flags & (1 << WATER_TEMPERATURE_BIT));
      Temperature2Valid = (Flags & (1 << TEMPERATURE_2_BIT));
      Temperature3Valid = (Flags & (1 << TEMPERATURE_3_BIT));
      WaterSpeedValid = (Flags & (1 << WATER_SPEED_BIT));
      PositionValid = (Flags & (1 << POSITION_BIT));
      DepthValid = (Flags & (1 << INVALID_DEPTH_BIT)) == 0;
      SurfaceValid = (Flags & (1 << SURFACE_DEPTH_BIT));
      TopOfBottomValid = (Flags & (1 << TOP_OF_BOTTOM_DEPTH_BIT));
      ColumnIs50kHz = (Flags & (1 << CHART_IS_50KHZ));
      TimeIsValid = (Flags & (1 << TIME_BIT));
      SpeedTrackValid = (Flags & (1 << EXTENDED_GPS_INFO_BIT));
#pragma warning(default : 4800)

      // Allow older version to ignore time indexing...
      if(TimeIsValid)
            NumberOfFish--;

      if(SpeedTrackValid)
            NumberOfFish -= 3;

      if(NumberOfFish < 0)
            return -1;

      if(UpperLimitValid)
      {
            UpperLimit = *(float *)(pData + Offset);
            Offset += sizeof(UpperLimit);
      }
      else
      {
            UpperLimit = 0.0F;
      }

      if(WaterTemperatureValid)
      {
            WaterTemperature = *(float *)(pData + Offset);
            Offset += sizeof(WaterTemperature);
      }
      else
      {
            WaterTemperature = 0;
      }
      if(Temperature2Valid)
      {
            Temperature2 = *(float *)(pData + Offset);
            Offset += sizeof(Temperature2);
      }
      else
      {
            Temperature2 = 0;
      }
      if(Temperature3Valid)
      {
            Temperature3 = *(float *)(pData + Offset);
            Offset += sizeof(Temperature3);
      }
      else
      {
            Temperature3 = 0;
```

```
        }
        if(WaterSpeedValid)
        {
                WaterSpeed = *(float *)(pData + Offset);
                Offset += sizeof(WaterSpeed);
        }
        else
        {
                WaterSpeed = 0;
        }
        if(PositionValid)
        {
                PositionY = *(long *)(pData + Offset);
                Offset += sizeof(PositionX);
                PositionX = *(long *)(pData + Offset);
                Offset += sizeof(PositionY);
        }
        else
        {
                PositionY = 0;
                PositionX = 0;
        }
        if(SurfaceValid)
        {
                SurfaceDepth = *(float *)(pData + Offset);
                Offset += sizeof(SurfaceDepth);
        }
        else
        {
                SurfaceDepth = 0;
        }
        if(TopOfBottomValid)
        {
                TopOfBottomDepth = *(float *)(pData + Offset);
                Offset += sizeof(TopOfBottomDepth);
        }
        else
        {
                TopOfBottomDepth = 0;
        }

        for(short FishIndex = 0; FishIndex < NumberOfFish; FishIndex++)
        {
                FishDepths[FishIndex] = *(float *)(pData + Offset);
                Offset += sizeof(FishDepths[FishIndex]);
        }

        if(TimeIsValid)
        {
                TimeIndex = *(long *)(pData + Offset);
                Offset += sizeof(TimeIndex);
        }
        else
        {
                TimeIndex = 0;
        }

        if(SpeedTrackValid)
        {
                Speed = *(float *)(pData + Offset);
                Offset += sizeof(Speed);
                Track = *(float *)(pData + Offset);
                Offset += sizeof(Track);
```

```
            Altitude = *(float *)(pData + Offset);
            Offset += sizeof(Altitude);
            if(Altitude == INVALID_ELEVATION)
            {
                  AltitudeValid = false;
                  Altitude = 0;
            }
            else
            {
                  AltitudeValid = true;
            }
      }
      else
      {
            Speed = 0;
            Track = 0;
            Altitude = 0;
            AltitudeValid = false;
      }

      return Offset;
}
```

E. Lowrance Mercator Meters

The position format is the same we have used since mapping was started and because of historical data saved and maps crea
correct radius cannot easily be changed without breaking everything. So we have our own unique position format.

```
//! The radius of the earth used for mercator meter conversions (JB: This is a polar
not equitorial radius)
#define MM_EARTH_RADIUS 6356752.3142

//! The maximum value of latitude for mercator meters
#define MAX_MM_LAT 0x3FFFFFFFL

//! The minimum value of latitude for mercator meters
#define MIN_MM_LAT -0x3FFFFFFFL

//! Circumference of earth in meters around the equator (JB: No, this is around the
poles.)
#define EARTH_CIRC 39940650

//! Longitude of international date line (180°) in mercator meters
#define DATE_LINE 19970325

RadiansLongitude = PositionX / MM_EARTH_RADIUS;
RadiansLatitude = ( 2.0 * atan( exp( PositionY / MM_EARTH_RADIUS ) ) ) - ( PI / 2.0 );
```