SL2

SL2 is a binary format used by Lowrance's (http://www.lowrance.com/) chart plotters to save tracks. This page describes the file structures as far as known. Of course, the format is implemented in Lowrance's tools which can be downloaded on their web site.

Inhaltsverzeichnis

File examples

```
0e3600584035dbf7a2cc013e85bf6ce209fffc21  DemoChart.sl2:  Primary,200kHz; Downscan,200kHz
e81ea02ac7797f16706632c662d41ac24a918a64  row.sl2:        Primary,200kHz; Downscan,800kHz; Sidescan,800kHz
c31a601462191c61725c05f8d24d2ab85435cbf2  sonar1.sl2:     Primary,200kHz; Downscan,800kHz; Sidescan,800kHz
bbbb950f4a06d251e6abdd7a4e575919e614ffff  sonarfresh.sl2: Primary,200kHz; Downscan,200kHz; Sidescan,200kHz
a35a0639499077fe7e1a4dd45e2dfa02204d2f9f  sonarsalt.sl2:  Primary,200kHz; Downscan,800kHz; Sidescan,800kHz
bd96f95e9a9ca5bda41607045580057064f9f851  sonar.slg:
a30f667d7dbbc7afdec82fc477cc76de7b5cad37  sonar_som.slg
```

Mark/Elite filename structure

TODO.

```
Chart 06_29_2015 [0].sl2
Chart [0].sl2
```

Basic Structure

The file is a binary file with little endian format. Float values are stored as IEEE 754 floating-point "single format".

The files show up with a 10 byte header. First 2 bytes describe the format version (01=SLG, 02=SL2). Bytes 5,4 provide the block size. It varies depending on the sensor: 0x07b2 for HDI=Primary/Secondary+DSI and 0x0c80 for Sidescan). Seen values are

```
               0  1  2  3  4  5  6  7  8  9                    
                                           Filesize    %144
            ? 02 00 01 00 b2 07 00 00 ?? ??   ?           ?
 DemoChart.sl2 02 00 00 00 b2 07 00 00 08 00    7032056  104
```

```
        row.sl2 02 00 00 00 80 0c 00 00 08 00   14289080  104
     sonar1.sl2 02 00 00 00 80 0c 00 00 08 00   16442888  104
 sonarfresh.sl2 02 00 00 00 80 0c 00 00 08 00   35607032    8
  sonarsalt.sl2 02 00 00 00 80 0c 00 00 08 00   25251272    8
      sonar.slg 01 00 00 00 92 09 00 00 00 00    4008210  114
  sonar_som.slg 01 00 00 00 80 0c 00 00 00 00   39811210  106
```

Then a sequence of blocks follows.

Starting from byte 0 the block contains:

variable type	length	offset	description
short unsigned n1	2 bytes	0	byte offset of frame = n1 + 65536 * n2
short unsigned n2	2 bytes	2	byte offset of frame = n1 + 65536 * n2
short unsigned n3	2 bytes		byte offset of last frame = n3 + 65536 * n4
short unsigned n4	2 bytes		byte offset of last frame = n3 + 65536 * n4
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short unsigned n5	2 bytes		byte offset of last frame = n5 + 65536 * n6
short unsigned n6	2 bytes		byte offset of last frame = n5 + 65536 * n6
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short?	2 bytes		Unknown (0)
short int	2 bytes	28	blockSize in bytes
short int	2 bytes	30	lastBlocksize of block (b - 1) in bytes
short int	2 bytes	32	Channel (sonar sensor type) ▪ 0 = Primary (Traditional Sonar) ▪ 1 = Secondary (Traditional Sonar) ▪ 2 = DSI (Downscan Imaging) ▪ 3 = Sidescan Left ▪ 4 = Sidescan Right ▪ 5 = Sidescan (Composite) ▪ Another other value is treated as Invalid.
short int	2 bytes	34	Packet size: Size of sounding/bounce data in bytes.
int	4 bytes	36	frameindex : used to identify which sensor recordings belong together starting at 0 and counting up

float	4 bytes	40	UpperLimit [feet]. Divide by 3.2808399 to get meters.
float	4 bytes	44	LowerLimit [feet]. Divide by 3.2808399 to get meters.
int	4 bytes	48	Unknown (0)
byte (int)	1 byte	52	Unknown
byte (int)	1 byte	53	Frequency ▪ 0 = 200KHz ▪ 1 = 50KHz ▪ 2 = 83KHz ▪ 3 = 455KHz ▪ 4 = 800KHz ▪ 5 = 38KHz ▪ 6 = 28KHz ▪ 7 = 130KHz - 210KHz ▪ 8 = 90KHz - 150KHz ▪ 9 = 40KHz - 60KHz ▪ 10 = 25KHz - 45KHz ▪ Any other value is treated as 200KHz
short int?	2 bytes	54	Unknown
int	4 bytes	56	time1 : first value contains ms since 1970 if multiplied by 1000, consecutive values have time encoded somehow different
float	4 bytes	60	WaterDepth [feet]. Divide by 3.2808399 to get meters.
int ?	4 bytes	64	Unknown
short ?	2 bytes	68	Unknown (0)
int ?	4 bytes	70	Unknown (seen values 71 and 100)
short	2 bytes	74	Unknown, 1000 seen often
float ?	4 bytes	76	Unknown, rarely appearing, -1 or values around 9.0 - 25.0
float ?	4 bytes	80	Unknown, rarely appearing, -1 or values around 9.0 - 25.0
float ?	4 bytes	84	Unknown, always -1
float ?	4 bytes	88	Unknown, always -1
short ?	2 bytes	92	Unknown, 0 , 1, 2, 3 or 257. could be a bitmask. sometimes occurs with index 76
short ?	2 bytes	94	Unknown
float	4 bytes	96	Speed [knots]. Divide by 1.94385 to get m/s . Based on GPS NMEA.
float	4 bytes	100	WaterTemperature [Celsius].
int	4 bytes	104	Easting [mercator meters] (Position X). See section after table for calculations.
int	4 bytes	108	Northing [mercator meters] (Position Y) See section after table for calculations.

float	4 bytes	112	WaterSpeed [knots]. Divide by 1.94385 to get m/s . This value is taken from an actual Water Speed Sensor (such as a paddlewheel). If such a sensor is not present, it takes the value from "Speed" (GPS NMEA data) and sets WaterSpeedValid to false.
float	4 bytes	116	Track/Course-Over-Ground in radians. Taken from GPS NMEA data.
float	4 bytes	120	Altitude [feet]. Divide by 3.2808399 to get meters. Taken from GPS NMEA data.
float	4 bytes	124	Heading in radians. Taken from GPS NMEA data.
short	2 bytes	128	Bitmask. From the left most bit: I don't think this is correct ▪ 0 = TrackValid ▪ 1 = WaterSpeedValid ▪ 2 = Unknown ▪ 3 = PositionValid ▪ 4 = Unknown ▪ 5 = WaterTempValid ▪ 6 = SpeedValid ▪ 7 to 13 = Unknown ▪ 14 = AltitudeValid ▪ 15 = HeadingValid
short int ?	2 bytes	130	Unknown
int ?	4 bytes	132	Unknown
int	4 bytes	136	TimeOffset. Time given in unknown resolution from unknown epoch. Unlike time1, this field is reliably populated. May be time of day in seconds (TOD, NMEA value) or time of the week (TOW, unlikely)
int	packetSize	140	Contains sounding/bounce data.

This should close the 144 byte frame.

Missing fields that are known to exist:

- DepthValid boolean

Notes:

- There does not appear to be a field for stating the device/GPS-antenna height to the surface of the water or to the transducer.
- KeelDepth directly affects WaterDepth. WaterDepth is thus logged with values after taking KeepDepth into account.
- Unknown values could include other GPS data or features that are specific to some models.
- The unit for water temperature is celcius but it is unknown if this can be changed in the logged file.

Latitude and Longitude

SL2 format stores Easting and Northing coordinates in Spherical Mercator Projection, using

WGS84 POLAR Earth radius (and NOT the WGS84 EQUATORIAL Earth Radius, as used by OpenStreetMap and Googel)

```
+proj=merc +a=6356752.3142 +b=6356752.3142
```

Easting values can be converted to WGS84 longitude by

```
POLAR_EARTH_RADIUS = 6356752.3142;

longitude = Easting / POLAR_EARTH_RADIUS * (180/M_PI);
```

Northing values can be converted to WGS84 latitude by

```
POLAR_EARTH_RADIUS = 6356752.3142;
double temp = Northing / POLAR_EARTH_RADIUS;
temp = exp(temp);
temp = (2*atan(temp))-(M_PI/2);

latitude = temp * (180/M_PI);
```

Implementations

- Java - sourceforge.net/projects/seasea (http://sourceforge.net/projects/seesea/) look under code for net.sf.seesea.navigation.sl2/src/net/sf/seesea/navigation/sl2/SL2Reader.java (http://sourceforge.net/p/seesea/code/HEAD/tree/trunk/net.sf.seesea.navigation.sl2/src/net/sf/seesea/navigation/sl2/SL2Reader.java) AND https://svn.code.sf.net/p/seesea/code/trunk/net.sf.seesea.navigation.sl2.test/src/net/sf/seesea/navigation/sl2/test/D.java
- Node.js - github/kmpm/node-sl2format (https://github.com/kmpm/node-sl2format)
- Golang - github.com/delitrem/sonarlog (https://github.com/delitrem/sonarlog) (changed license to more permissive one)
- R - [1] (https://stackoverflow.com/a/52282967/1457051)
- Vala - github.com/delitrem/echogram (https://github.com/delitrem/echogram) (incomplete sl2 viewer for GTK+)
