Navico (Lowrance, Simrad, B&G)

Sonar Log File Format

Herbert Oppmann

herby@memotech.franken.de

http://www.memotech.franken.de/FileFormats/

2021-06-02

Content

Navico Sonar Log File Format

Filename extensions *.slg (format 1, sonar only), *.sl2 (format 2, sonar and structure) and *.sl3 (format 3, includes ForwardScan)

This documentation is based on own research and the sources listed in the references section.

The tool Insight Map Creator (previously Navico MapCreate) is able to read *.slg and *.sl2 files and convert it to a *.shp (ESRI Shapefile), *.lsf (Lowrance Shapefile) or *.csv (comma-separated-values) file from that. To do that, select View > Processing Modes > Sonar Mode Window. With *.shp only Latitude and Longitude are converted. With *.lsf and *.csv, depth, channel, quality, and speed are also converted.

The tool Lowrance Sonar Log Viewer is also able to read *.slg and *.sl2 files and convert it to a *.csv (comma-separated-values) file. But it converts much more data fields and is therefore preferrable.

Text highlighted like this is where further investigation is needed.

Basic data types

All values are serialized in little-endian byte order (least significant byte first).

Type	Length	Description
byte	1	8 bit unsigned integer (range 0 .. 255)
ushort	2	16 bit unsigned integer (range 0 .. 65535)
uint	4	32 bit unsigned integer (range 0 .. 4294967295)
short	2	16 bit signed integer (range -32768 .. 32767)
int	4	32 bit signed integer (range -2147483648 .. 2147483647)
float	4	32 bit floating point value according to [18].

Mercator meter to WGS84 conversion (in C#):

```
private const double PolarEarthRadius = 6356752.3142;
private const double RadiansToDegrees = 180.0 / Math.PI;

public static double Longitude(int Easting)
{
   return (Easting / PolarEarthRadius) * RadiansToDegrees;
}

public static double Latitude(int Northing)
{
    return ((2.0 * Math.Atan(Math.Exp(Northing / PolarEarthRadius))) - (Math.PI / 2.0)) * RadiansToDegrees;
}
```

General file structure

Header
List of Frames (no gaps) The last frame sometimes has incomplete sounding / bounce data.

Header

Offset	Type	Content
+0	ushort	Format 1 = *.slg, 2 = *.sl2, 3 = *.sl3
+2	ushort	Version 0 = ex HDS 7 1 = ex Elite 4 CHIP 2 = ?
+4	ushort	BytesPerSounding, see table below.
+6	byte	Debug. 0= off, <>0 (usally 1) on If set, enables the ChannelTypes Debug Digital and Debug Noise.
+7	byte	=0
	Only if Format =1:	
+8	ushort	=0

BytesPerSounding seen values:

Value	Meaning
0x0064 (100)	? Not seen in Format 1
0x04B0 (1200)	? Only in Format 1
0x0640 (1600)	Seen in Format 3
0x07B2 (1970)	Downscan, seen in Format 1+2
0x0992 (2450)	? Only in Format 1
0x0C80 (3200)	Sidescan, seen in Format 1+2+3

According to screenshot from a device, also 200, 400, 800, 2000 and 2400 can be set.

Frame for Format 1

See source code embedded in [10] and file addon\src\main\java\org\vaadin\sonarwidget\data\LowranceSonar.java in [11].

The header has variable size. Some fields are always present, even if there exists some flag indicating the validity of the field value. Other fields, marked with “[Cond]” for “conditional”, are only present when the corresponding flag bit indicates “valid”. To further complicate things, conditional fields which are present because the corresponding “valid” flag is set may have a special value indicating “invalid”!

Type	Content
ushort	Flags, see table below
float	LowerLimit Lower depth limit of the transducer in use
float	WaterDepth in feet. There is no specific “invalid” value here when the DepthInvalid flag is set.
float	[Cond] UpperLimit
float	[Cond] Water temperature in °Celsius
float	[Cond] WaterSpeed Speed through the water from the paddlewheel speed sensor. Actual water speed (or GPS if sensor not present) in knots.
uint	[Cond] Northing in mercator meters

Type	Content
uint	[Cond] Easting in mercator meters
float	[Cond] Surface depth in feet
float	[Cond] TopOfBottom depth in feet
float	[Cond] Temperature 2 in °Celsius
float	[Cond] Temperature 3 in °Celsius
float	[Cond] Present if Unknown Flag0 is set AND SpeedAndTrackValue is set This is some depth in feet, similar to Depth and Top of bottom depth, but different than Surface depth.
2 x float	[Cond] Present if Unknown Flag1 is set. The two values are similar to each other, but not similar to any other values here.
float	TimeOffset in ms from the start of the log
float	[Cond] Speed (GPS) Speed from GPS in knots.
float	[Cond] Track in radians
float	[Cond] Altitude in feet above sea level A value of -10000.0 is considered “invalid”.
ushort	PacketSize. Seen values: 1148 or 1152 (when BytesPerSounding =1200), 1920 (when BytesPerSounding =1970), 2400 (when BytesPerSounding =2450), 3072 (when BytesPerSounding =3200)
byte[n]	Sounding / bounce data (no specific structure, just a list of values) A little bit shorter than PacketSize. Filled with 0x00 bytes until BytesPerSounding bytes are full.

Flag bits for Format 1:

Bit offset / mask	Meaning if set
0 0x0001	Unknown Flag0
1 0x0002	Unknown Flag1
2 0x0004	Altitude valid
3 0x0008	Upper Limit valid
4 0x0010	Water temperature valid
5 0x0020	Temp2 valid
6 0x0040	Temp3 valid
7 0x0080	Water speed valid
8 0x0100	Position valid (fields Northing and Easting)
9 0x0200	Depth **in**valid
10 0x0400	Surface depth valid
11 0x0800	TopOfBottom depth valid
12 0x1000	ColumnIs50kHz
13 0x2000	TimeOffset valid
14 0x4000	Speed and track valid
15 0x8000	Unused, =0

Frame for Format 2

The size of the frame header is 144 bytes. In contrast to Format 1, the fields are always present, regardless of the “valid” flags. In other words, the header has a fixed layout.

General info on “Last *type* channel frame offset in file”: The current frame already counts, i.e. if the current frame is of *type*, the value is the same as “Frame offset in file”. If there was no *type* frame yet, the value is 0.

Offset	Type	Content
+0	uint	Frame offset in file In first frame = 8.
+4	uint	Last primary channel frame offset in file
+8	uint	Last secondary channel frame offset in file
+12	uint	Last downscan channel frame offset in file
+16	uint	Last left sidescan channel frame offset in file
+20	uint	Last right sidescan channel frame offset in file
+24	uint	Last composite sidescan channel frame offset in file
+28	ushort	FrameSize This is the actual size of the frame in bytes. It is 94 or 144 more than the BytesPer-Sounding given in the header(?) Seen values: 2064 (when BytesPerSounding =1970) 656, 1344, 1544, 2144, 2544, 2944, 3024 or 3216 (when BytesPerSounding =3200)
+30	ushort	PreviousFrameSize Size of the previous frame (FrameIndex-1) in bytes. In first frame =0.
+32	ushort	ChannelType, see table below.
+34	ushort	PacketSize = 144 less than FrameSize.
+36	uint	FrameIndex Starts with 0 and counts up. The counting is done separately for all frames of each ChannelType.
+40	float	UpperLimit in feet Upper depth limit of the transducer in use
+44	float	LowerLimit in feet Lower depth limit of the transducer in use
+48	ushort	Unknown2. Seen values: 0, 12, 20, 21, 27, 50, 130, 520, 4956, 5000, 5050, 5100, 5150, 5200, ...
+50	byte	Unknown3. Seen values: 0, 4, 5, 8, 13, 25, 32, 40, 48, 56, 64, ...
+51	byte	Unknown4. Seen values: 0, 19, 41, 43, 46, 47, 48, 49, 50, 79, 80, ...
+52	byte	Unknown5. Seen value: 0, 1
+53	byte	Frequency, see table below.
+54	ushort	Unknown6. Seen values: 0, 41, 48, 49, 51, 53, 100, 767, 793, 795, 806, 854, ...
+56	ushort	Unknown7. Seen values: 0, 1, 8, 16, 24, 32, 45, 48, 71, 72, 88, 104, 112, 156, 160, 192, 224, 232, 240, 244, 257, 513, 514 Maybe flags or separate bytes
+58	ushort	Unknown8. Seen values: 0, 1
+60	int	CreationDateTime. A value of -1 means not set. Otherwise it is a POSIX time, see [19].
+64	float	WaterDepth in feet
+68	float	KeelDepth in feet
+72	byte	Unknown9. Seen values: 0x00-0x02, 0x04, 0x07, 0x09, 0x0C, 0x0E, 0x10-0x5A, 0x5D, 0x5F, 0x62, 0x64, 0x66, 0x69, 0x6B, 0x6D, 0x6E, 0x70, 0x73, 0x75, 0x78, 0x7A, 0x7C, 0x7F, 0x81, 0x84, 0x86, 0x89, 0x8B, 0x8E, 0x90, 0x93, 0x95, 0x97, 0x9A, 0x9C, 0x9F, 0xA1, 0xA4, 0xA6, 0xA9, 0xAB, 0xB0, 0xB2 Values are increasing or decreasing in consecutive frames.

Offset	Type	Content
+73	byte	Unknown10. Seen values: 0x00, 0x04, 0x06-0x0B, 0x0D-0x15, 0x17-0x25, 0x28, 0x2A, 0x2C-0x30, 0x32-0x46, 0x48-0x4E, 0x50-0x55, 0x57, 0x58, 0x5B-0xE6 Values are increasing or decreasing in consecutive frames.
+74	ushort	Unknown11. Seen values: 0x0000, 0x1305, 0x2805, 0x6205, 0x6505, 0x735D, 0x7454, 0x7457, 0x745A, 0x745D, 0x7460 and many many more. Maybe two separate bytes.
+76	byte	Unknown12. Seen values: practically all byte values
+77	byte	Unknown13. Seen values: 0, 1, 30, 60, 70, 80 and 120
+78	ushort	Unknown14. Seen values: 0, 1, 100, 1000
+80	float	Unknown15. Seen value: -1.0, ..
+84	float	Unknown16. Seen value: -1.0, ..
+88	float	Unknown17. Seen value: -1.0, ..
+92	float	Unknown18. Seen value: -1.0
+96	byte	Unknown19. Seen values: 0, 1, 2, 3
+97	byte	Unknown20. Seen values: 0, 1, 2, 3
+98	byte	Unknown21. Seen values: 0, 1, 2, 3
+99	byte	Unknown22. Seen values: 0, 1
+100	float	Speed (GPS) Speed from GPS in knots.
+104	float	Water temperature in °Celsius
+108	int	Easting in mercator meters
+112	int	Northing in mercator meters
+116	float	WaterSpeed Speed through the water from the paddlewheel speed sensor. Actual water speed (or GPS if sensor not present) in knots.
+120	float	Track (course over ground) in radians
+124	float	Altitude in feet above sea level
+128	float	Heading in radians
+132	ushort	Flags, see table below
+134	ushort	Unknown23. Seen value: 0
+136	byte	Unknown24. Seen values: 1, 3
+137	byte	Unknown25. Seen values: 0, 1, 2, 4, and 6
+138	byte	Unknown26. Seen values: 10, 13, 15 and 20 Note these are often multiples of 5
+139	byte	Unknown27. Seen values: 0, 1, 2 and 3
+140	uint	TimeOffset in ms from the start of the log
+144	byte[PacketSize]	Sounding / bounce data (no specific structure, just a list of values)

Flag bits for Format 2:

Bit offset / mask	Meaning if set
0 0x0001	Unknown, not seen
1 0x0002	GPS speed valid
2 0x0004	Water temperature valid
3 0x0008	Unknown, but seen
4 0x0010	Position valid
5 0x0020	Unknown, but seen

Bit offset / mask	Meaning if set
6 0x0040	Water speed valid
7 0x0080	Track valid
8 0x0100	Heading valid
9 0x0200	Altitude valid
10-15	Unknown, not seen

Depth is always valid.

Frame for Format 3

The size of the frame header is 128 bytes for ChannelType 7 and 8, else 168 bytes.

Offset	Type	Content
+0	uint	Frame offset in file In first frame = 8.
+4	uint	Seen value: 10
+8	ushort	FrameSize This is the actual size of the frame in bytes.
+10	ushort	PreviousFrameSize Size of the previous frame (FrameIndex-1) in bytes. In first frame =0.
+12	ushort	ChannelType, see table below.
+14	ushort	Seen value: 0
+16	uint	FrameIndex Starts with 0 and counts up.
+20	float	UpperLimit in feet Upper depth limit of the transducer in use
+24	float	LowerLimit in feet Lower depth limit of the transducer in use
+28	...	Seen bytes: 00 00 05 00 00 00 00 00 30 00 00 00, 19 00 04 32 00 04 28 00 00 00 00 00, 45 00 04 11 00 02 28 00 00 00 00 00
+40	uint	CreationDateTime. A value of -1 means not set. Otherwise it is a POSIX time, see [19].
+44	ushort	PacketSize Size of the sounding/bounce data in bytes. It should be 144 less than FrameSize.
+46	ushort	Seen value: 0
+48	float	WaterDepth in feet
+52	byte	Frequency, see table below.
+53	...	Seen bytes: 00 00 00 44 B8 0F F8 E5 1E E8 03 The first four could be float 512.0 but: FA A7 3F 33 00 00 00 00 01 01 00 does not look like float
+64	float	Unknown15. Seen values: -1.0, 21.74922, 22.45391, ...
+68	float	Unknown16. Seen values: -1.0, 21.62109, ...
+72	float	Unknown17. Seen value: -1.0, 21.0125, ...
+76	float	Unknown18. Seen value: -1.0
+80	byte	Unknown19. Seen value: 0, 1
+81	byte	Unknown20. Seen value: 0, 1
+82	byte	Unknown21. Seen value: 0
+83	byte	Unknown22. Seen value: 0

Offset	Type	Content
+84	float	SpeedGPS Speed from GPS in knots.
+88	float	Water temperature in °Celsius
+92	int	Easting in mercator meters
+96	int	Northing in mercator meters
+100	uint	WaterSpeed Speed through the water from the paddlewheel speed sensor, measured in knots. It is zero if no paddle wheel water speed sensor is attached.
+104	float	Track (course over ground) in radians
+108	float	Altitude in feet above sea level
+112	float	Heading in radians
+116	uint	Seen values: 438, 446, 690, 698, 946, 950, 952, 954, 958 In one file, there are always two values with 8 apart.
+120	byte	Unknown24. Seen value: 1
+121	byte	Unknown25. Seen values: 0, 1, 4, and 6 (maybe 0=Depth invalid?)
+122	byte	Unknown26. Seen values: 0, 10 and 20 Note these are multiples of 5
+123	byte	Unknown27. Seen values: 1 and 2
+124	uint	TimeOffset in ms from the start of the log
+128	uint	Last primary channel frame offset in file
+132	uint	Last secondary channel frame offset in file
+136	uint	Last downscan channel frame offset in file
+140	uint	Last left sidescan channel frame offset in file
+144	uint	Last right sidescan channel frame offset in file
+148	uint	Last composite sidescan channel frame offset in file
+152	uint[3]	Seen values: 0
+164	uint	Last 3D scan channel frame offset in file
+168		

Sounding / bounce data if ChannelType = 0 (Primary) or 7:

Sequence of bytes, starting with 99

Sounding / bounce data if ChannelType = 8:

Sequence of ushort, with values around 0x0750

Sounding / bounce data if ChannelType = 9 (3D):

Offset	Type	Content
+0	uint	Size of header in byte = 76
+4	uint	S1 = Size of Section 1 in byte (may be 0!)
+8	uint	S2 = Size of Section 2 in byte (may be 0!)
+12	uint	S3 =Size of Section 3 in byte = S3a + S3b
+16	uint	S3a = Size of Subsection 3a in byte
+20	uint	S3b = Size of Subsection 3b in byte
+24	uint[7]	=0
+52	float	?
+56	uint[5]	=0

Offset	Type	Content
+76	byte[S1]	Section 1 (see below)
...	byte[S2]	Section 2 (see below)
...	byte[S3a]	Section 3a (see below)
...	byte[S3b]	Section 3b (see below)
...	0-3 byte	Alignment to 4-byte boundary, =0

The sum

- Size of Frame header (168) +
- Size of Sounding data header (76) +
- Size of Section 1 +
- Size of Section 2 +
- Size of Section 3 +
- alignment bytes

must be the FrameSize.

Section 1 and 2

Type	Content
(S1 or S2)/8 times pair of float:	
float	Integer number, ascending (but not continuous), starting with 0
float	Value

Section 3a and 3b

Type	Content
uint	? The first two uints and the next two uints are similar. Coordinates?
uint	?
uint	?
uint	?
uint[(S3a or S3b-16-5)/4]	? uint or float?
byte[4]	?
byte	? Seen value: 2

ChannelType values:

In Format 1 only values 0-5 seen.

Value	Meaning
0	Primary (Traditional Sonar)
1	Secondary (Traditional Sonar)
2	DSI (DownScan imaging)
3	Left (SideScan)
4	Right (SideScan)
5	Composite (SideScan)
7	? (maybe also: only if Debug on)
8	? (maybe also: only if Debug on)

Value	Meaning
9	3D (only in Format 3)
10	Debug Digital (only if Debug on)
11	Debug Noise (only if Debug on)
other	invalid

Frequency values:

Value	Meaning
0	200 kHz
1	50 kHz
2	83 kHz
3	455 kHz
4	800 kHz
5	38 kHz
6	28 kHz
7	130-210 kHz
8	90-150 kHz
9	40-60 kHz
10	25-45 kHz
160	? 83/800 kHz
other	treated as 200 kHz

References

Used sources of information

[1] OpenStreetMap Wiki http://wiki.openstreetmap.org/wiki/SL2

[2] See Sea - Java Libraries for Maritime Navigation http://sourceforge.net/projects/seesea/

[3] Module to read .sl2 files https://github.com/kmpm/node-sl2format

[4] SL2 library for Go https://web.archive.org/web/20180613011000/github.com/delitrem/sonarlog

[5] Small C# API for working with sonar log files https://github.com/risty/SonarLogApi

[6] Ruby code https://github.com/Chris78/sl2decode

[7] Tools to Read 'Lowrance' Binary Track Files https://gitlab.com/hrbrmstr/arabia

[8] Sounder Log Formats https://github.com/opensounder/sounder-log-formats

[9] Convert Lowrence SL2 files to streaming/newline-delimited JSON (ndjson) https://github.com/hrbrmstr/sl2json

[10] Geotech forum thread “Lowrance MCC saved data structure” http://www.geotech1.com/forums/showthread.php?11159-Lowrance-MCC-saved-data-structure

[11] Vaadin add-on for viewing sonar logs https://github.com/capeisti/SonarWidget

[12] Console application to cut .sl2 files https://github.com/Volccc/sl2cut

[13] Decoding sonar sl2 files https://github.com/nwhoffman/sl2PyDecode

[14] https://github.com/SPoslavskyi/snr2dem

[15] Snapper https://github.com/nickazg/Snapper

[16] sonar https://github.com/karlkastner/sonar

[17] SL3Reader https://github.com/halmaia/SL3Reader

Standards and specifications

[18] ISO/IEC/IEEE 60559:2011 Information technology – Microprocessor Systems – Floating-Point arithmetic (which is the successor of IEEE Std 754-2008)

[19] Unix time https://en.wikipedia.org/wiki/Unix_time

Sources of sample files

[20] Sonar0???.sl2 from http://www.lochnessinvestigation.com/castlecruisessonar.htm

[21] https://downloads.lowrance.com/software/index.html?r=798 → ftp://software.lowrance.com/HDS_4.1.36.68_with_Simulator_Files.zip

[22] ftp://software.lowrance.com/Simulators/Lowrance-Inland-Simulator.sl3

[23] Data\sonar1.sl2 in installation directory of Sonar Viewer 2.1.2
http://www.navionika.com/media/emulators/SonarViewer_2.1.2.exe

[24] http://www.navionika.com/media/emulators/sonar_som.slg

[25] inst\extdat\example.sl2 in https://gitlab.com/hrbrmstr/arabia

[26] test\fixtures*.sl2 in https://github.com/kmpm/node-sl2format

[27] testdata\chart.sl2 in https://github.com/hrbrmstr/sl2json

[28] Sonar0015.sl2 in https://github.com/nwhoffman/sl2PyDecode

[29] sonar_logs\Sonar000?.sl2 in https://github.com/nickazg/Snapper

[30] Nunit.Tests\input.sl? In https://github.com/risty/SonarLogApi