Genesis Log File Format Specification (*.glf files)

0716-SDS-00001

Table of Figures

List of Tables

1. Amendment Details

1.1. Version History

Rev	***Date***	***Superseded Documents / Description / Details***
01	07/02/2020	First Release (KM/AS)
02	19/05/2020	Gemini Ping flag definition updated, added GPS structure, added routine to calculate range in meters (AS)
03	06/07/2021	Added compression type filed for Gemini data structure and source name for sensor structure. Added Altimeter and Bathy structure.
04	20/10/2021	Added Gemini status message structure and fixed typos.
05	18/02/2022	Added support for Micron Gemini and fixed Gemini Acoustic Zoom structure
06	21/11/2022	GImage (Table 10) version 2 update details, Gemini Status Record (Table 11) add Internal Pressure/Temp Sensor and IMU Sensor details.
07	31/08/2023	Updates for Generic Header, SeaKing records and Gemini GImage

2. Introduction

2.1. Purpose

The intention of this document is to define the specification for the proprietary Tritech Log File format that is used in the Genesis software application. These files have the **".glf"** file extension.

This document is intended for software developers and other parties that have an interest in its contents.

2.2. Scope

This specification applies to an electronic Log File format that is specific and proprietary to Tritech. It is designed to be an extensible format that easily supports the addition of new sensor data types as well as having the ability to be editable and changeable without affecting data replay.

This specification and the new .glf log format follow other proprietary Log File formats written by Tritech and used in their other software applications, namely Seanet Pro (.V4Log log format) and Gemini software (.ecd log format).

The ".glf" log files are designed to be cross-platform so they can be opened in other, non-Windows®, operating systems such as Linux.

The format can also be applied to non-Tritech applications (e.g. other than Genesis) so users can open and playback data from ".glf" files in their own software applications.

2.3. Benefits and Objectives

As aforementioned, this log file specification has been written to be cross platform compatible, not tying it down to use on specific platforms and with specific classes and libraries.

It is designed to be flexible and allow for expansion as new product models, lines and functions are included in future Genesis software application upgrades.

2.4. **Definitions and abbreviations**

The definitions, abbreviations and acronyms used in this Specifications document are listed below.

- **ADCP:** Acoustic Doppler Current Profiler. A hydroacoustic current meter used to measure water current velocities over a depth range using the doppler effect of sound waves scattered back from particles within the water column.
- **ASCII:** Abbreviation for American Standard Code for Information Interchange which is a character encoded standard for electronic communication.
- **AVI**: Audio Video Interleave. A format of multimedia file used in many computer systems for video/audio recording and playback.
- **DST**: Stands for "Digital Sonar Technology". A range of Sonar products designed by the Company that includes new FPGA based architecture and has the capability of Chirp transmissions.
- **.ECD**: A format of log file, proprietary to Tritech, that stores multibeam and ancillary data into a file structure for later playback within the legacy standalone Gemini software application.
- **FPGA:** Field Programmable Gate Array. Is a semiconductor device, or integrated circuit, that has an array of programmable logic blocks which can be programmed to perform complex functions with an in-built block of memory allowing data and program storage.
- **Genesis:** A Tritech software suite to control, log and display data from its range of subsea devices and sensors along with incorporation of 3rd party sensors.
- **.GLF**: The file extension of the Genesis Log File format that is specific to the Genesis software application and is a proprietary format of the Company.
- **GMT:** Greenwich Mean Time is the mean solar time at the Royal Observatory in Greenwich, London. Was used as the international civil time standard until replaced by UTC.
- **GPS**: Global Positioning System.
- **Linux**: Linux is an Open Source computer operating system designed primarily for the PC but also available for a wide range of other systems.
- **MRU:** Motion Reference Unit. A device installed with multi-axis sensors to measure motion.
- **SCADA:** A Supervisory Control and Data Acquisition system that as its name suggests is an application which controls and archives data from a range of connected devices.
- **SDK**: Software Development Kit. This typically comprises a set of reference documents, software libraries & source code examples provided to the customer so

they can communicate with a particular item of hardware directly through the development of their own supervisory software application.

- **User**: The person or persons who will operate and interact directly with the Mayan SDK and its components.
- **UTC:** The Universal Time Coordinated system of time. Is the primary time standard used to regulate and synchronize all clocks and times across the planet. It is determined using highly precise atomic clocks and the earth's rotation.
- **UTM:** Universal Transverse Mercator. Is a co-ordinate system that divides the earth into sixty Zones of 6° bands which defines Longitude (e.g. East to West). Latitude bands (e.g. North to South) are segmented into 8° degree high sections and are allocated letters, with are sequenced alphabetically between "C" and "X", with "O" & "I" excluded; Latitude bands "A", "B", "Y" and "Z" cover the Antartic and Artic regions. The combination of a Zone and Latitude band defines a grid zone marking a specific area location on the planet.
- **V4:** The communications protocol used in a range of products designed by the Company, including SeaKing, SeaPrince and Micron.
- **.V4Log:** A format of log file, proprietary to Tritech, used in the Company's Seanet Pro software application.
- **WMV**: Windows Media Video. A Microsoft developed format of multimedia file used in Windows® computer systems for video/audio recording and playback.

2.5. References

[1] 0685-SDS-00002, Gemini Interface Specification.

[2] "SeaNet_DigSon_Protocol.doc" Notes for use of V4 protocol to interface to a DST Sonar.

3. Record Types

The ".glf" log file is a compressed archive file and in Genesis will contain two main sub files: **.cfg** and **.dat**. Other supporting files will be contained in the .glf in later Genesis version such as .xml *(for storing logging, capture & casting settings and desktop settings)* and also chart bitmaps.

- The **.cfg** file is formatted as an XML file and contains a range of different fields and records that fall under several categories:

  Log File Header
  Copy of Project Settings
  Device Configuration Records
  Log File Terminator

  The .**dat** file stores all the sensor data and sensor information records:

  1) **Sensor/Device Records**
  2) **System/Information Records**

3.1. CFG File Contents

The Record structures have their own unique header sections followed by data/information payloads. Each will now be explained in addition to the Log File Header / Terminator and Project Settings capture.

For user applications that generate .glf files for playback in Genesis, create a .cfg with populated Log File Header and Log File Terminator sections (i.e. main XML elements) then create a Project Settings section but leave its XML element empty. Genesis will then open the .glf log file in the project that it has currently opened.
A section with the user application project settings or data can be added as a section between the Log File Header and the Log File Terminator. As long as the XML element tag name used is unique (e.g. it can use an XML Namespace if necessary) then it will be ignored by Genesis.

3.1.1. **Log File Header**

The header element includes identifiers and version numbers in addition to storage of the list of devices that were enabled during recording. A load code is allocated so that future modes of operation can be fixed in playback. A file open date/time is also stored.

The **Log File Header** has the following structure:

Data Type	Variable Name	Default Value	Description
unsigned int	m_logStart	FDFEFEFFh	Start of file Sequence Value = FDFEFEFFh always *(decimal = 4261347071)*
unsigned short	m_fileVersion	1	A constant which identifies the Log File version. Hi Byte = Major, Lo Byte = Minor. e.g. x.x
std::string	m_appName	"Genesis"	This is a string that identifies the name of the Software Application that generated the ".glf" log file during playback processing. *(The Genesis version number will be appended in later Genesis versions)*
double	m_creationTime	0	Log File open Date / Time in local time with millisecond resolution. This is the creation Date / Time.
unsigned char	m_loadCode	0	This byte code will normally be set to 0 during recording. Once recording is complete and the ".glf" log file is closed off, the file can be edited later and this value can be changed to then be able to open the file in Genesis under another specific mode. *(Currently 0 Always = Open Normally)*
std::string	m_deviceInfo	""	For Genesis this is a comma-separated string list that contains all the Devices present in the log file. The Application can therefore read this list to determine what devices to add into the application for the data playback. This field can also be edited to only create specific devices in the application to limit the log playback only to those devices. Example: "*Gemini – 901, MicronNav – 90*". …where 901 and 90 are the ID/Node numbers. **Note:** Currently defaults to "No-devices"
std::string	m_userInformation	""	Multi-purpose field where user can enter any text information such as a log.

Table 1 – Log File Header Structure

The Log File Header record is stored under the ‘**<logHeader>**’ section (i.e. XML element) in the “.cfg” log file.

3.1.2. **Copy of Project Settings**

When a new log recording is opened, the current “.gns” project file that’s used in Genesis is stored in the “.cfg” log file. This will allow the exact same layout to be re-created for the data playback that was evident at time of recording.

The “.gns” project file is in XML format and all its contents fall between the ‘<GuiCurrentSettings>’ and ‘</GuiCurrentSettings>’ start and end tags, e.g.

```
<GuiCurrentSettings>
      <application>
      .
      .
      .
      </application>
      <devices>
      .
      .
      .
      </devices>
      .
      .
      .
      etc
</GuiCurrentSettings>
```

The above XML project settings are stored under the ‘**<GuiCurrentSettings>**’ section (i.e. XML element) in the “.cfg” log file.

3.1.3. **Device Configuration Records.**

These are device records which include device configuration parameters and firmware information. These are not runtime configuration settings such as Range Scale and Gain for a Sonar for instance, but version numbers, port configurations and such like which are configured offline.

For instance, devices that use the V4 protocol such as SeaKing/Micron/SeaPrince Sonar heads will send **Version** and **UserBB** (User Boot Block) configuration details to the application during startup. The ‘VersionInfo’ and ‘UserCfg’ portions of these messages can be logged. These messages will only be sent to the application once (unless re-powered) and during the initialisation sequence before they commence ping/sample operations.
Device configuration records from different types of device, e.g. V4 Protocol based (such as SeaKing) and Gemini, will have different structures. Supported structures are detailed below.

All applicable device configuration records are stored under the ‘**<deviceConfigs>**’ section within the “.cfg” log file.

‘V4 Protocol’ Device Version and UserBB Records

These configuration records will be in the following format:

1. Version Information

mtVersionData (from Node 2)

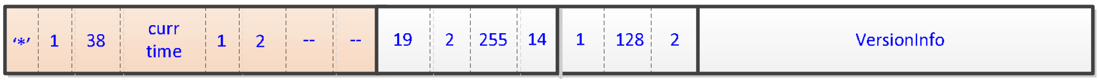

Figure 1 – Example of V4 VersionInfo Device Configuration Record Structure

2. User Boot Configuration Information

mtBBUserData (from Node 2)

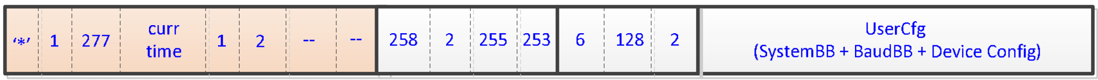

Figure 2 – Example of V4 UserCfg Device Configuration Record Structure

Gemini Sonar License Information

This is licensing information stored inside the Gemini device that for instance can instruct the software application to enable a particular licensed feature such as Target Tracking.

This configuration structure is programmed into the flash memory of the Gemini device using an encoded format. This contains the Gemini device’s unique ID number which is matched against the ID stored with the device’s recorded ping data records in the log file. Therefore, for security purposes it is not possible to copy any license structure into a log file and use this with Gemini devices that have other unique IDs. The ID in the license structure must match the ID in the recorded ping data.

3.1.4. Log File Terminator

This Record signifies the end of the “.glf” log file. It has an End of Log identifier and a file close time.

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned int	m_logEnd	FFFEFEFDh	End of file Sequence Value = FFFEFEFDh always *(decimal = 4294901501)*
double	m_closeTime	0	Log File close Date / Time in UTC with millisecond resolution (local time). This is the file close Date / Time at time of recording only.

Table 2 – Log File Terminator Structure

3.2. **DAT File Contents**

Important: All records stored in the ".dat" file will be written in Little-endian format (i.e. Intel 80x86 notation).

3.2.1. **Sensor/Device Records**

The ".dat" Log File will support recording and playback of a number of Sensor and Device data records & input strings, including:

1) Gemini Sonar *(e.g. Imager models)*
2) Video frames *(e.g. camera input)*
3) Target Tracking related records
4) Micron/SeaPrince/SeaKing Sonar
5) MicronNav
6) GPS NMEA strings
7) Gyro and Magnetic Compass
8) MRU
9) Altimeter
10) Bathy

This list will increase as the Genesis software application expands in the future.

The Sensor/Device Records will be interspersed with any System Information Records (*see Section 3.2.2*).

All Sensor/Device Records will consist of a Common Interface Header followed by a data payload. The data payload may include device data from Proprietary V4 Protocol or Gemini Tritech devices or it may provide storage of data from a number of Generic 3rd party devices such as GPS and Compass.

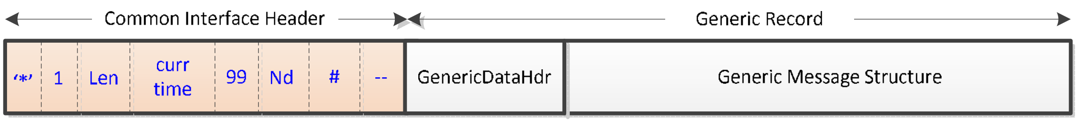

Figure 3 – Example of Sensor/Device Record Structure

The **Common Interface (CI) Header** has the following structure:

Field	**Bytes**	**Definition**
m_idChar	1	Start character used to signify start of CI header (Always = "*")
m_version	1	Version number of the CI Header
m_length	4	Total length (number of bytes) of CI Header and Data message *(Note: will be 0 for Video Records)*
m_timestamp	8	Local Time given as seconds from 1 Jan 1980 with millisecond

		resolution (double value, i.e. milliseconds after decimal point)
m_dataType	1	What type of data is contained in attached data record? 0 = SVS5 (e.g. Gemini) 1 = V4 Protocol (e.g. SeaKing, SeaPrince, Micron) 2 = Analog_Video 3 = Gemini Status . . 50 = Marker Input (Chart/Nav) * 60 = System Event (Errors and Notifications) * 70 = Bookmark * 98 = Raw_Serial 99 = Generic *= Not currently supported*
m_deviceID	2	This will be: - A Gemini Sonar ID number when m_dataType = 0 - Ignore when m_dataType = 1, i.e. m_nodeID gives V4 device ID number, For m_dataType = 99, this field then becomes the Device Instance indicator (e.g. 1st GPS: *m_deviceID* = 1, 2nd GPS: *m_deviceID* = 2, etc)if pseudo Node number is allocated in Genesis software then write this number in, else ID = 0
m_nodeID	2	This will be: - A pseudo Node ID number allocated in Genesis software when m_dataType = 0, 3 or 99: Gemini = 100..109 Gemini Aux Port 0 = Gemini Node ID + 10, Gemini Aux Port 1 = Gemini Node ID + 10 + 1 Compass = 228, 246 (NB Sonar Aux = 232) GPS = 245 Altimeter = 226 (NB Sonar Aux = 231) - The Node ID of the V4 device, when m_dataType = 1: SeaKing/SeaPrince/Micron Sonar = 2..9 SeaKing Bathy = 40..44 MicronNav = 90 MicronNav Attitude Sensor = 75..76
m_spare	2	Reserved for future expansion

Table 3 – CI Header Structure

The **GenericDataHdr** has the following structure:

Field	Bytes	Definition
m_version	2	Version number *(to track structure changes)*
m_len	2	Length of the Generic Message Structure / data record that follows this header.

m_genType	1	Generic Data Type, is an index that maps into an Enumeration, e.g.
		`enum EGenericType {`
		`GT_RECORD_RAW_ASCII_STRING, // 0`
		`GT_RECORD_GPS,`
		`GT_RECORD_COMPASS,`
		`GT_RECORD_RAW_V4_REPLY,`
		`GT_RECORD_PROC_V4_REPLY,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_BAROMETER,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_ALTIMETER`
		`GT_RECORD_DEPTH,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_BATHY,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_RESERVED,`
		`GT_RECORD_RAWAHRS,`
		`GT_RECORD_ALTIMETER_EXT };`

Table 4 – GenericDataHdr Record

The GenericDataHdr includes the Generic Type (m_genType) for the input data and the Generic Message Structure contains its raw ASCII string received into Genesis.

Gemini Sonar Records

There are a few record types that will be stored for the Gemini when it is operating. These are Target Image Records, Ping Tail Extension Records and Acoustic Zoom Records.

Gemini Target Image Records

The Gemini Sonar ping data is stored in ".glf" formatted Log Files as a Target Image Record with the Common Interface Header preceding it.

The full record structure which is stored is as follows:

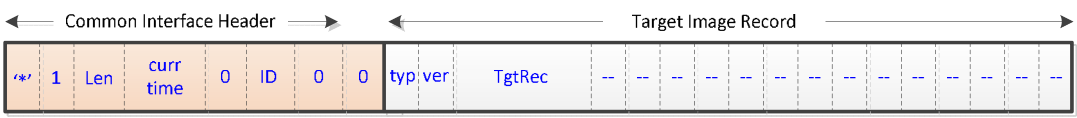

Figure 4 – Example of Genesis Log Target Image Record Structure

The **Genesis Log Target Image Record** represents a combined data format that maintains all the Gemini sonar acquisition data and parameters. A Target Image Record contains all the data for 1 ping, which corresponds to one image, or frame, on the sonar display. The structure is as follows:

Data Type	**Variable Name**	**Description**

CiHeader	m_ciHeader	Common interface header (See above for details)
RecHeader	m_recHeader	Header contains type and tag code (See below for details)
GMainImage / GProfiler	m_mainImage	Main image of sonar. (See below for details)
GZoomImage	m_aczImage	Acoustic Zoom image if enabled (See below for details)
EndTag	m_usEndTag	Const value (0xDEDE)

Table 5 – Genesis Log Target Image Record

The **Record Header structure** represents record type of record saved including version information

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned short	Type		This is a constant which identifies the Record Type, e.g. 0 = Undefined Record Type 1 = Genesis Image Record
unsigned short	Version		This is a constant which identifies the versioning of this Record Type, e.g. 0xEFEF = Genesis Image Record

Table 6 – Genesis Record Header Type

The **GMainImage structure** encapsulate GImage and other fields to represents the whole frame for 1 ping. The structure is as follows:

Data Type	**Variable Name**	**Default Value**	**Description**
GImage	m_image	See Gemini Image Record *(GImage)* details below	
std::vector<double>	m_vecBearingTable	Size of bearing table can be calculated (m_uiEndBearing - m_uiStartBearing)	
unsigned int	m_uiStateFlags	0x0000	State Flags. Bit 13-15 sonar orientation

unsigned int	m_uiModulationFrequency	40000	The modulation frequency. This is used to calculate the range scale. See section 5 Appendix
float	m_fBeamFormAperture	120	Calculate the number of beams to provide the correct aperture.
double	m_dbTxTime	0	The time of transmit.
unsigned short	m_usPingFlags	0	The ping flags returned in the ping.[1] **Bit 8: (1:HF, 0:LF)** **Bit 15: (1: Manual, 0: Sonar) SOS**
float	m_fSosAtXd	1500	The speed of sound of the transducer.
signed short	m_sPercentGain	0	The percentage gain used in the recording.
bool	m_fChirp	0	True = Chirp, False = Continuous Wave.
unsigned char	m_ucSonarType	0	Sonar type selected (Imager or Profiler) Imager=0 Profiler=1 (Not supported in GLF) Mk2 Image = 2
unsigned char	m_ucPlatform	0	none=0 720is=1 720ik=2 720im= Micron Gemini = 3 1200ik=4

Table 7 – Genesis Main Image Record (GMainImage)

(Note: This device is not currently supported in Genesis…)

The **GProfilerImage structure** encapsulate GMainImage and other fields to represents the whole frame for 1 ping. The structure is as follows:

Data Type	Variable Name	Default Value	Description
GMainImage	m_profiler		See Gemini Image Record (GImage) details below
unsigned char	m_ucStartApertureBeamIndex	0	**Profiler data only**: corresponds to beam number where start aperture is located, e.g. -65° ≈ beam 0.
unsigned char	m_ucStopApertureBeamIndex	255	**Profiler data only**: corresponds to beam number where stop aperture is located, e.g. 65° ≈ beam 255.

float	m_fRollFromDMG	0	**Profiler data only**: roll value from MRU device communicating with Gemini standalone legacy software.
unsigned char	m_ucStartApertureBeamIndex	0	**Profiler data only**: corresponds to beam number where start aperture is located, e.g. -65° ≈ beam 0.
unsigned char	m_ucRollCompensationStatus	0	**Profiler data only**: has roll compensation

Table 8 – Genesis Profiler Main Image Record (GProfilerImage)

Certain Gemini Imager models (e.g. MK2) have an Acoustic Zoom feature. When enabled, Acoustic Zoom Records will also start to be broadcast and logged in the ".dat". This represents a combined data format that maintains all the sonar zoom acquisition data and parameters. It will contain all the data for 1 ping which is one zoom image frame on the sonar display screen.

There will be one **Acoustic Zoom Record** for every ping and these records will also follow the Genesis Main Image Records. The structure is as follows:

Data Type	Variable Name	Default Value	Description
bool	m_fActive	false	Zoom Enabled. (***Important**: m_usId & m_dMagnitude fields and the m_image will only be appended if this flag is enabled*)
unsigned short	m_usId	0	Acoustic zoom ID
double	m_dMagnitude	1.5	Default scaled value
GImage	m_image	See Gemini Image Record (GImage) details below	

Table 9 – Genesis Zoom Image Record (GZoomImage)

The **Gemini Image (GImage)** contains the raw ping data for one Image Frame. Its structure is as follows:

Data Type	Variable Name	Default Value	Description
unsigned short	m_usImageVersion	3	Version info to keep track changes

unsigned int	m_uiStartRange	0	The start range (range line #)
unsigned int	m_uiEndRange	0	The end range (range line #)
unsigned short	m_usRangeCompUsed	0	The range compression used for this ping The value returned from the ping head
unsigned int	m_uiStartBearing	0	The start bearing (beam #)
unsigned int	m_uiEndBearing	512	The end bearing (beam #)
unsigned short	m_usCompressionType	0	0: zLib compression, 1: Raw data 2: H264 compressed data Supported in version 3 or above
unsigned int	m_uiDataSize	0	Vector data size.
std::vector<unsigned char>	m_vecData	0	Vector array of size uiDataSize. This data may be compressed using one of type mentioned above in m_usCompressionType

Table 10 – Gemini Image (GImage)

Gemini Sonar Status Records

The Gemini sonar sends status message to the host which contains important information about the sonar e.g. the firmware version and temperature of different hardware components etc. The GLF logger now supports to log Gemini status messages, so that the exact environment can be reproduced during the playback. Also, sometimes it helps debugging the user's issues.

The full record structure which is stored is as follows:

Figure 5 – Example of Gemini Sonar Status Record Structure

The **Gemini Sonar Status Record** contains all the data for sonar status. The status record contents will vary according to Gemini model – see **0685-SDS-00002** for Gemini model specifics. An example of the structure is as follows:

Data Type	Variable Name	Default Value	Description
CiHeader	m_ciHeader	Common interface header (See above for details)	
unsigned short	m_statusMsgVersion	1	Status Message Version

unsigned short	m_bfVersion	0	BF version Info
unsigned short	m_daVer	0	DA version info
unsigned short	m_flags	0	Internal use
unsigned short	m_deviceID	0	Sonar device ID
unsigned char	RESERVED	0xFF	Reserved for Internal Usage
double	m_vgaT1	0	MK2: FPGA PCB temperature
double	m_vgaT2	0	MK2: HSC PCB temperature
double	m_vgaT3	0	MK2: DA FPGA
double	m_vgaT4	0	Transducer temperature
double	m_psuT	0	PSU temperature
double	m_dieT	0	Die temperature
double	m_txT	0	TX temperature
double	m_afe0TopTemp	0	AFE0 Top temperature
double	m_afe0BotTemp	0	AFE0 Bottom temperature
double	m_afe1TopTemp	0	AFE1 Top temperature
double	m_afe1BotTemp	0	AFE1 Bottom temperature
double	m_afe2TopTemp	0	AFE2 Top temperature
double	m_afe2BotTemp	0	AFE2 Bottom temperature
double	m_afe3TopTemp	0	AFE3 Top temperature For Micron Gemini, is used to send Ext. Water Temp Val when m_dipSwitch bit 1 = 1, e.g. *degC = TempVal – 2000 / 100* (Note: TempVal = 0xFFFF = invalid)
double	m_afe3BotTemp	0	AFE3 Bottom temperature For Micron Gemini, is used to send Ext. Pressure Val when m_dipSwitch bit 1 = 1, e.g. *bar = (300000 / 2^16) * (PressVal / 10000)* (Note: PressVal = 0xFFFF = invalid)

unsigned short	m_linkType	0	Bit 0 – link type 0 – Ethernet 1 – VDSL Bit 3..1 (Relevant only for VDSL) Bit 9..8 – link speed (Ethernet) 00 – 10 Mbps 01 – 100 Mbps 10 – 1000 Mbps Bit 11..10 – 720im/Micron Gemini link type 00 - No link (broadcast) 01 - Ethernet 10 - Serial RS232 11 – Serial RS485 Bit 12 Negotiation status 0 – Negotiating 1– Negotiation completed Bit 13 – Network adaptor status 0 – Network adaptor not found 1 – Network adaptor found
double	m_uplinkSpeedMbps	0	Up Link Speed
double	m_downlinkSpeedMbps	0	Down Link Speed
unsigned short	m_linkQuality	0	Link quality in percentage
unsigned int	m_packetCount	0	Packet counts (both Tx and Rx) for the device
unsigned int	m_recvErrorCount	0	Received error count
unsigned int	m_resentPacketCount	0	Resent packet Count
unsigned int	m_droppedPacketCount	0	Dropped packet Count
unsigned int	m_unknownPacketCount	0	NOT USED
unsigned int	m_lostLineCount	0	Lost line count for device
unsigned int	m_generalCount	0	Packet count for the devices attached
unsigned int	m_sonarAltIp	0	Alternate IP Address
unsigned int	m_surfaceIp	0	Currently connected surface PC IP address
unsigned int	m_subNetMask	0x00ffffff	Subnet mask of the sonar
unsigned short	m_macAddress1	0	First 2 bytes of Mac Address

unsigned short	m_macAddress2	0	Middle 2 bytes of Mac Address
unsigned short	m_macAddress3	0	Last 2 bytes of Mac Address
unsigned int	m_BOOTSTSRegister	0	INITERNAL USAGE
unsigned int	m_BOOTSTSRegisterDA	0	INITERNAL USAGE
uint64_t	m_fpgaTime	0	64 bit timestamp (micro time)
unsigned short	m_dipSwitch	0	INTERNAL USAGE
unsigned short	m_shutdownStatus	0	Bit 0 – Over Temperature shutdown Bit 1 – Out of water shutdown Bit 2 – Out of Water indicator
bool	m_networkAdaptorFound	false	(m_linkType & 0x2000) ? true : false
double	m_subSeaInternalTemp	0	For DMD: SubSea Internal temperature
double	m_subSeaCpuTemp	0	For DMD: SubSea CPU temperature

Table 11 – Gemini Sonar Status Record

Video Records

The camera frame image along with any future optional audio are logged in the ".dat".

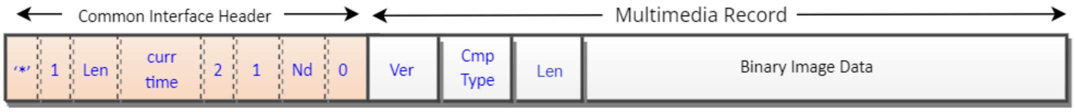

Figure 6 – Example of Multimedia Record Structure (Video)

There will be one **Multimedia Record** for every video frame captured from the camera device. The structure is as follows:

Data Type	Variable Name	Default Value	Description
unsigned short	m_imageVersion	1	Frame Index of captured image (= last video Frame idx if audio or text)
unsigned int	m_compressionType	0xFFFFFFFF	Always 0xFFFFFFFF
unsigned int	m_uiLength		The length *(number of bytes)* that follow for the Binary Image Data in a video capture (or Audio/Text data for other multimedia types)
(Compressed) Image Format or audio/text format	Binary Image Data / Audio Data / Text data		e.g. Video Image data is ZLib compressed and is OpenCV BGR byte-ordered.

Table 12 – Multimedia Record

Target Tracking records

Target Tracking is available on the Gemini Imager models. These have several data records that are stored during Log recording:

Target Position

These are computed position outputs from the Target Tracking feature, e.g. 2 string formats are output that are stored during Log recording:

- ✓ $PTRITAR (created for click 'n' track)
- ✓ $PTRITR1 (created for detect 'n' track)

Note: Currently the above option is not supported in the .glf.

Tidal Flow Vector

This is the input from an ADCP over SCADA for instance. It is used in the target tracking as an aid to distinguish drifting targets from live targets and also used in swim velocity calculations for example.

This data input is handled and processed in Genesis as Generic Data (*see Table 4 – GenericDataHdr Record, for structure breakdown*). The data is logged as Generic Records with a Common Interface Header, e.g.

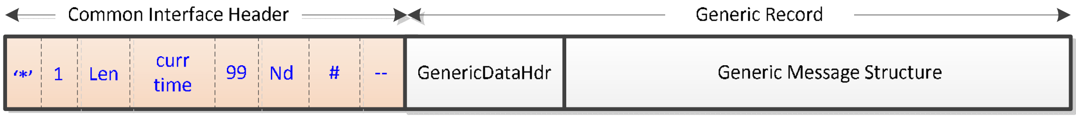

Figure 7 – Example of Generic Record Structure

The Global node of 244 is set in the Common Interface Header which means any device in Genesis can access and process this data. The ADCP string is stored in the Generic Data Record as follows:

Figure 8 – Example of ADCP data stored in Generic Record Structure

Note: Currently the above option is under development so not yet supported in the .glf.
(The '99' = Generic may also be changed to '98' = Raw Serial when feature is implemented)

Micron/SeaPrince/SeaKing Sonar records

These Mechanical Sonar models all use the V4 communications protocol (refer to "SeaNet_DigSon_Protocol.pdf" notes).

There are several common device reply packets that each of these Sonar models transmit which are stored in the log file, e.g.

- mtHeadData
- mtVersionData
- mtBBUserData

The **mtVersionData** and **mtBBUserData** are only transmitted during power-up initialisation. These device packets contain programmed device parameters and information which may later be stored once in the ".glf" – i.e. their content will not change during operation. Details for storage of these records is contained in Table 7.

The **mtHeadData** device packets contain sample data for 1 ping (= 1 image scanline). They are the device reply packets for each Sonar ping. Also contained in these device packets is a copy of the device configuration parameters (Range, Gain etc) for the current ping.

All device reply packets will be stored with the following format:

Figure 9 – Example of stored V4 Device reply packet Record Structure

So, the mtHeadData packets are stored in the ".dat" log file with a Common Interface Header applied, which is the same as for other Company devices such as Gemini *(see details on storage of Gemini Target Image Records earlier in this Section).*

The full record structure for the mtHeadData storage is as follows:

Figure 10 – Example of mtHeadData storage Record Structure

Here, the **Device Data** structure forms the Device Message Structure portion of the storage record. This Device Data structure contains a copy of the configuration parameters applied to the ping in addition to all the sample data for the ping (i.e. producing *x* number of A/D "Bins" for 1 scanline). The Device Data structure is as follows:

Data Type	Variable Name	Default Value	Description
DeviceParams	m_devParams	See Device Parameters Record (DeviceParams) below.	
ReplyData	m_repData	0x000C	Array of samples/bins making up the scanline. Read ‘m_dataBytes’ in the preceding Device Parameters Record to determine the quantity of sample data for the current ping/scanline. Each sample/bin is either a 4-bit or 8-bit value… If the data is in packed 4-bit mode then there will be twice as many bins as Data Bytes (m_dataBytes). For 8-bit mode, then bins = Data Bytes. For 4-bit and 8-bit bins, the Sonar will always return an EVEN number of range BINS, even if an Odd number has been requested. Odd numbers will be rounded up to the nearest Even value. If adc8on = 0 in HdCtrl then the 4-bit data bins are packed as follows: BYTE45 BYTE46 BYTE47 …… BIT 7654 3210 7654 3210 7654 3210 …… BIN 0000 1111 2222 3333 4444 5555 …… If adc8on = 1 then the 8-bit data bins are packed as follows: BYTE45 BYTE46 BYTE47 …… BIT 7654 3210 7654 3210 7654 3210 …… BIN 0000 0000 1111 1111 2222 2222 ……

Table 13 – Device Data Record (params/samples from mtHeadData)

For the Mechanical Sonar, its **Device Parameters Record** block is as follows:

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned short	m_byteCount	0	Total Byte Count of Device Parameters + Scanline Sample Data (“Bins”). ***Note****: In Multi-Packet mode is total across all packets.*
unsigned char	m_devType	11	Device Type (02 = CW Sonar, 11 = DST Sonar)
unsigned char	m_hdStatus	0	Head Status byte (bitset) where: ***bit 0*** *: ‘HdPwrLoss’ (Head in reset condition)* ***bit 1*** *: ‘MotErr’ (Motor lost sync, reboot)* ***bit 2,3*** *: Always 0 (‘PrfSyncErr’,’PrfPingErr’)* ***bit 4*** *: In 8-bit ADC mode.* ***bit 5,6*** *: Reserved.* ***bit 7*** *: Message is Appended.*
unsigned char	m_swpCode	0	Sweep Code value where: ***‘0’*** *= Scanning Normal* ***‘1’*** *= Scan At Left Limit* ***‘2’*** *= Scan At Right Limit* ***‘3’*** *= Reserved* ***‘4’*** *= Scan At Centre/Ahead Position*

unsigned short	m_hdCtrl	0	Head Controls byte (2-byte bitset) where: ***bit 0*** *: 'adc8on' ()* ***bit 1*** *: 'cont' ()* ***bit 2*** *: 'scanright' ()* ***bit 3*** *: 'invert' ()* ***bit 4*** *: 'motoff' ()* ***bit 5*** *: 'txoff' ()* ***bit 6*** *: 'spare' ()* ***bit 7*** *: 'chan2' ()* ***bit 8*** *: 'raw' ()* ***bit 9*** *: 'hasmot' ()* ***bit 10*** *: 'applyoffset' ()* ***bit 11*** *: 'pingpong' ()* ***bit 12*** *: 'stareLLim' ()* ***bit 13*** *: 'ReplyASL' ()* ***bit 14*** *: 'ReplyThr' ()* ***bit 15*** *: 'IgnoreSensor' ()*
unsigned short	m_rngScale	100	The low order 14 bits are set to a value of Rangescale * 10 units. **Bit 6**, **Bit 7** of Byte 15 are used as a code (0..3) for the Range Scale units, where: **0** = Metres, **1** = Feet, **2** = Fathoms, **3** = Yards. E.g. 20 metre range : Range Scale = 200 20 Yard range : Range Scale = 200 + 2^14 + 2^15
unsigned int	m_txN	0	Transmitter constant for current operating channel (e.g. Low Freq or High Freq for SeaKing). ***Note****: Not used for DST heads, so set to zero.* F = Transmitter Frequency in Hertz TXN (chan) = F * 2^32 / 32e6 RXN (chan) = (F + 455000) * 2^32 / 32e6 E.g. chan1 = 325 kHz, chan2 = 675 kHz TXN (chan1) BYTE 4,5,6,7 = 43620761 TXN (chan2) BYTE 8,9,10,11 = 90596966 RXN (chan1) BYTE 12,13,14,15 = 104689827 RXN (chan2) BYTE 16,17,18,19 = 151666032

unsigned char	m_gain	105	Initial Gain of the Receiver for the current operating channel. (Gain value 0..210 = 0..+80dB)
unsigned short	m_slope	90 / 125 *(for 325 / 675kHz Sonar)*	A Slope value which affects the Time Variable Gain (TVG) operation within the sonar head. The TVG compensates for spreading and attenuation losses of sound through water. Typical values… No TVG Default Slope 0 200kHz Default Slope 70 325 kHz Default Slope 90 580 kHz Default Slope 110 675 kHz Default Slope 125 795 kHz Default Slope 130 935 kHz Default Slope 140 1210 kHz Default Slope 150 2000 kHz Default Slope 180
unsigned char	m_adSpan	38	'ADSpan' and 'ADLow' control the mapping of the received sonar echo amplitudes. The sonar receiver has an 80dB dynamic range, and signal levels are processed internally by the sonar head such that 0..80dB = 0..255. **ADSpan** (m_adSpan) is the window size.
unsigned char	m_adLow	40	The **ADLow** (m_adLow) field determines the base position of the display dynamic range data in the 80dB sonar data window, and effectively provides some control over the sensitivity of the sonar.
unsigned short	m_hdgOffset	0	Heading Offset for compass device. ***Note**: This offset is used with Sonar models that have the compass option as a special configuration*
unsigned short	m_adInterval		Defines the sampling interval of each Range Bin. It is in units of 640 nanoseconds. Its use is tightly coupled with the Number of Bins. The maximum Number of Bins in SeaKing/SeaPrince/Micron Sonar heads is 800. In the newer digital DST range of Sonar heads, the maximum is increased to 1500. A practical minimum for ADInterval is about 5 (approx. 3 microseconds). The ADInterval may also be rounded slightly to fit in with Receiver calculations (i.e. a request of 14830 may be adjusted to 14829 in the Sonar to fit into whole sampling periods for the required Range Scale.

unsigned short	m_leftAngleLim	0	'm_leftAngleLim' and 'm_rightAngleLim' determine the scanning sector of the Sonar head. Once set, the Sonar head will automatically steer to the next direction within these limits whenever a new 'mtSendData' ping interrogate command is received, and will automatically reverse scan direction at these scan limits if the 'cont' bit in 'm_hdCtrl' is clear. The current units for these values are in 1/16th Gradian units, and are limited to the range 0..6399. The SeaKing direction convention is as follows: Left 90° = 1600 Ahead = 3200 Right 90° = 4800 Astern = 0 (or 6399)
unsigned short	m_rightAngleLim	6399	See above.
unsigned char	m_stepAngle	8	This byte sets the scanning motor step angle between 'pings' and is in units of 1/16 Gradians. The SeaKing default Resolution settings are: **Low** = 32 (= 2 Gradians = 1.8°) **Medium** = 16 (= 1 Gradian = 0.9°) **High** = 8 (= 0.5 Gradians = 0.45°) **Ultimate** = 4 (= 0.25 Gradians = 0.225°)
unsigned short	m_txdcrBrg	0	Transducer Bearing. This is the position of the Transducer for the current ping/scanline (0..6399 in 1/16 Gradian units).

unsigned short	m_dataBytes	0	Number of Data Bytes. This value is used to control reading the Sonar Scanline data Bins that follow. It defines the number of Range Bins that the sonar will generate for the Data reply message. The operating range of the Sonar is defined by the time taken to do 'Nbin' samples at an ADInterval such that the Scanline data time = 'Nbins' of ADInterval * 640nanoseconds. The maximum value of 'Nbins' is limited to 800 in SeaKing/SeaPrince/MiniKing/Micron heads and is limited to 1500 in newer digital 'DST' heads. The Sonar will always return an EVEN number of Bins, regardless of whether an odd number has been requested. For example, if 7 bins were requested then 8 bins would be sampled and returned. An odd number is always rounded up to the nearest even number.

Table 14 – Device Parameters Record (DeviceParams) for a Node 2 Sonar

Preceding the Device Data Record in the stored packet are the **LAN Packet Header** (LanPktHdr) and the **LAN Message Header** (LanMsgHdr). The format of the structures are as follows:

Data Type	Variable Name	Default Value	Description
unsigned short	m_binLen	0	Length of 'Device Data Record' (e.g. LanPktHdr + LanMsgHdr + Device Message Structure)
unsigned char	m_sid	2	Packet Source Identification (e.g. Tx Node number 0 – 240)
unsigned char	m_did	255	Packet Destination Identification (e.g. Rx Node number 0 – 255)
unsigned char	m_count	0	= 0 if Single Packet of Device Data (i.e. normally this field is Byte count of remaining message that follows this structure)

Table 15 – LAN Packet Header (LanPktHdr) Record

Data Type	Variable Name	Default Value	Description
unsigned char	m_msgType	2	Message Type. *where (main types listed only):* ***1** = mtVersionData,* ***2** = mtHeadData,* ***4** = mtAlive,* ***6** = mtBBUserData,* ***8** = mtAuxData* ***16** = mtReBoot* ***19** = mtHeadCommand* ***23** = mtSendVersion* ***24** = mtSendBBuser* ***25** = mtSendData*
	m_msgSeq	128	This is a Bitset that should be processed for Multi-packet modes in order to determine if the "mtHeadData" reply message exceeds 128 bytes and will therefore be split into sequential packets of less than 128 bytes in length. ***Note**: Should always = 80h for devices that do not use the Multi-packet mode.* The Message Sequence Bitset is constructed as follows; **Bits 0 – 6** : Packet Sequence bits. 1st packet has Packet Sequence of 0, 2nd packet has Packet Sequence of 1 with bit 0 = 1 (0000001), 3rd packet has Packet Sequence of 2 (0000010), 4th packet has Packet Sequence of 3 (0000011)…etc. **Bit 7** : End Sequence bit. Set to 1 when it is the last packet in sequence. For instance, the last packet in a 3-packet sequence will have a Bitset of 10000010 (=82H).
unsigned char	m_node	2	Tx Node Number (0 – 240)

Table 16 – LAN Message Header (LanMsgHdr) Record

MicronNav records

The MicronNav USBL system will generate several types of Record. It has 2 main sensors which are assigned Node numbers pre-allocated and fixed for the system:

- Node 90 - This is the Acoustic USBL component of the system which is used for ranging and positioning
- Node 75 or 76 - This the Heading, Pitch and Roll Sensor fitted inside the surface USBL transducer which produces attitude and bearing data to angularly correct the raw USBL position data, in the production of real-world co-ordinates.

Both Nodes communicate using the proprietary V4 communications protocol. Like Micron and SeaKing devices, they transmit Version and UserBB information on startup. However their raw data packets are processed before being logged so the format of recorded data does not directly follow that of other V4 devices (a header is different).

Both Nodes, 75 and 90 will transmit the following records in line with other V4 devices. These records will be stored in the log file, e.g.

- mtVersionData
- mtBBUserData

The **mtVersionData** and **mtBBUserData** are only transmitted during power-up initialisation. These device packets contain programmed device parameters and information which may be stored once in the ".glf" – i.e. their content will not change during operation. Details for storage of these records is contained in Table 7.

Node 75 will store the following 'processed' data record:

- CompassDataRecord

Node 90 will store the following 'processed' data record:

- AMNavData

A processed CompassDataRecord will be stored for Node 75, which (although data types remain the same) is not the same structure that is recorded for Gyro and Magnetic Compass devices. The Common Interface Header will indicate that it is Node 75 that is producing these records for the MicronNav system. The stored structure is follows:

Figure 11 – Example of Node 75 Compass Record Structure

The 'CompassDataRecord' payload in the above structure will be a structure as produced by the Application, containing motion and position data calculated from raw Node 75 MRU sensor data, e.g.

Data Type	Variable Name	Default Value	Description
double	m_heading	0	The Compass Heading in degrees
double	m_roll	0	The Roll angle in degrees
double	m_pitch	0	The Pitch angle in degrees
double	m_heave	0	Unused for Node 75
double	m_xValue	0	Unused for Node 75
double	m_yValue	0	Unused for Node 75
double	m_zValue	0	Unused for Node 75
double	m_temp	0	Unused for Node 75
double	m_variation	0	Unused for Node 75
double	m_deviation	0	Unused for Node 75
bool	m_hdgIsTrue	false	Always false for Node 75
unsigned int	m_cmpValid	0	32-bit Bitset, where: Bit 0 = 1 if Heading Valid Bit 1 = 1 if Pitch Valid Bit 2 = 1 if Roll Valid *(Other Bits may be set for use in Genesis and can be ignored)*
char	m_sourceName[16]	""	Name label for the Compass
unsigned long long	m_time	0	64-bit (milliseconds into day value)

Table 17 – CompassDataRecord storage Record

For Node 90, its calculated position data will be stored as 'processed' AMNavData records with a preceding Common Interface Header. The stored record will have the following structure:

Figure 12 – Example of Node 90 V4 Device data packet Record Structure

Note that this is different to the payload stored for other V4 devices such as SeaKing Sonar (*see*

Figure 9) where the initial header is not a LanPktHdr but instead a ConMsgHdr. The difference between the two headers is that the ConMsgHdr used by the Node 90 MicronNav has a Time value written into it. This is the ping time for the current USBL position.

So, the AMNavData records are stored in the ".glf" log file with a Common Interface Header applied, which is the same as for other Company devices such as Gemini *(see details on storage of Gemini Target Image Records in this Section).*

The full record structure for the AMNavData storage is as follows:

Figure 13 – Example of AMNavData storage Record Structure

Here, the **AMNavData** structure forms the Device Message Structure portion of the storage record. This AMNavData structure contains a copy of the configuration parameters applied to the ping in addition to the data associated with the ping. The AMNavData structure is as follows:

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned int	m_ping_mecs	0	The time of the ping in units of milliseconds into the day.
unsigned char	m_replies	0	Unused for now. Always 0.
float	m_doaX	0	Unused for now. Always 0.
float	m_doaY	0	Unused for now. Always 0.
float	m_doaZ	0	Unused for now. Always 0.
float	m_rms	0	Unused for now. Always 0.

float	m_usblAngQual	0	Unused for now. Always 0.
float	m_usblRngQual	0	Unused for now. Always 0.
float	m_reliabilityX	0	Unused for now. Always 0.
float	m_reliabilityY	0	Unused for now. Always 0.
float	m_reliabilityZ	0	Unused for now. Always 0.
float	m_reliabilityR	0	Unused for now. Always 0.
double	m_aAttitude[3]	0	Unused for now. Always 0.
bool	m_xpdrMode	0	Unused for now. Always 0.
unsigned short	m_unitID	0	0 = Responder ‘R0’, 1..15 = Transponder ‘T1’..’T15’
TP3D	m_relPos	[0,0,0]	Relative XYZ USBL calculated position.
float	m_sigmaX	0	Unused for now. Always 0.
float	m_sigmaY	0	Unused for now. Always 0.
float	m_sigmaZ	0	Unused for now. Always 0.
float	m_sigmaR	0	Unused for now. Always 0.
float	m_sigmaYX	0	Unused for now. Always 0.
float	m_fixVOS	1500	Velocity Of Sound applied to the calculated USBL Position (in metres/second)
double	m_range	0	Unused for now. Always 0.
TP3D	m_attCorrPos	[0,0,0]	m_relPos corrected for Pitch, Roll and Heading from Node 75 Compass Record. Is calculated XYZ relative position of the responder/transponder.
TP3D	m_worldPos	[0,0,0]	Absolute position of responder/transponder, referenced to world USBL transducer coordinates, given in Degrees Latitude and Longitude.
double	m_fixDateTime	0	The Date and Time of when the Ping fix occurred. Currently unused. Use Ping Time in ConMsgHdr.

Table 18 – AMNavData Record for a Node 90 MicronNav

where:

```
struct TP3D {           // 3D Cartesian Position(metres)
    double m_x;
    double m_y;
    double m_z;
};
```

Preceding the Device Data Record in the stored packet are the **CON Message Header** (ConMsgHdr) and the **LAN Message Header** (LanMsgHdr). The latter is described in Table *7*, whereas the ConMsgHdr will now be described:

Data Type	Variable Name	Default Value	Description
unsigned short	m_binLen	0	Length of 'Device Data Record' (e.g. LanPktHdr + LanMsgHdr + Device Message Structure)
double	m_pingTime	0	Time the acoustic ping was fired which is used in the attached AMNavData position record. Time is no of days passed since 12/30/1899 (fractional part is milliseconds into current day).
unsigned char	m_sid	90	Packet Source Identification (e.g. Tx Node number 0 – 240)
unsigned char	m_did	255	Packet Destination Identification (e.g. Rx Node number 0 – 255)

Table 19 – CON Message Header (ConMsgHdr) Record

GPS records

These records primarily contain NMEA data strings that are inserted into Generic Data Records before logging (*see Figure 3 – Example of Sensor/Device Record Structure' and Table 3 – CI Header Structure' in Section 3.2.1*).

Figure 14 – Example of GPS Record Structure

(Note: The '99' is stored as processed GPS records recorded in Genesis, '98' = Raw Serial in non-Genesis applications. This means it can be any string format and relies on the software decoding the log file to look for compatible string formats such as NMEA GGA/GLL/RMC)

Where the 'GpsDataRec' payload in above structure will be a generic structure specific to Genesis which has the following structure:

Data Type	Variable Name	Default Value	Description

LatLongPosition	m_llRec	0	structure with position in Degs & Mins format
EastingNorthingPosition	m_enRec	0	structure Easting/Northing (UTM/OSGB)
double	m_cog	0	Course Over Ground (Degrees)
double	m_sog	0	Speed Over Ground (Knots)
double	m_heading	0	Heading (Degrees)
SatelliteRecord	m_satInfo	0	Unused (Structure with Satellite Info)
unsigned long long (64bit)	m_time	0	Seconds from Jan 1, 1970 (Unix)
char	m_gpsValid	0	Bitset, where: Bit 0 = 1 if Time Valid Bit 1 = 1 if Heading Valid Bit 2 = 1 if SOG Valid Bit 3 = 1 if COG Valid Bit 4 = 1 if Lon Valid Bit 5 = 1 if Lat Valid
char	m_sourceName[16]	“”	Name label for the GPS
char	m_enFormat	‘’	unused if blank, ‘H’ Rel Heading up, ‘N’ Rel North up, ‘E’ Abs UTM

Table 20 – GpsDataRecord storage Record

Where LatLongPosition structure

Data Type	Variable Name	Default Value	Description
double	m_latDegrees	0	latitude in degrees
double	m_longDegrees	0	longitude in degrees
double	m_altitude	0	altitude in meters

EastingNorthing structure

Data Type	Variable Name	Default Value	Description

double	m_easting	0	Easting
double	m_northing	0	Northing
double	m_altitude	0	Altitude

SatelliteRecord structure

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned int	m_satsInView	0	total number of satellites
unsigned int	m_id[12]	0	IDs of up to 12 satellites used for fix
unsigned int	m_elev[12]	0	satellite elevations in degrees
unsigned int	m_azim[12]	0	satellite azimuth in degrees
unsigned int	m_snr[12]	0	satellite SNR 00-99
double	m_PDOP	0	Position Dilution of Precision
double	m_HDOP	0	Horizontal Dilution of Precision
double	m_VDOP	0	Vertical Dilution of Precision
int	m_mode2	0	Fix Type: 1 = N/A, 2 = 2D, 3 = 3D

Gyro / Magnetic Compass records

As with GPS records, these Compass records will also be logged within Generic Data Records. CompassDataRecord will be the raw data string as received from the compass. The logged record structure is as follows:

Figure 15 – Example of Compass Record Structure

(Note: The '99' is processed compass records recorded in Genesis, '98' = Raw Serial in non-Genesis applications. This means it can be any string format and relies on the software decoding the log file to look for compatible string formats such as NMEA HDT/HDM)

Where the 'CompassDataRecord' payload in above structure will be a generic structure specific to Genesis which has the structure defined in Table 17 – CompassDataRecord storage Record.

Motion Reference Unit / Attitude Sensor records

Is considered to be part of Compass processing, so structures in the Compass Record *section* will apply here.

Altimeter records

These records primarily contain altitude data strings from the PA Altimeter and MicronEchosounder that are inserted into Generic Data Records before logging (*see Figure 3 – Example of Sensor/Device Record Structure' and Table 3 – CI Header Structure' in Section 3.2.1*).

Figure 16 – Example of Altimeter ASCII Record Structure

(Note: The '99' is the processed Altimeter record (see table 12) recorded in Genesis, '98' is Generic Raw Serial)

Where the AltimeterRec in above structure will contain the altitude value (in metres) from the PA Altimeter, e.g.

Data Type	Variable Name	Default Value	Description
double	m_altitudeMetres	0	Altitude in meters
double	m_vosUsed	0	The velocity of sound
bool	m_altValid	0	True if valid else false
double	m_slantRangeMetres	0	if PAPA-AHRS then we have AHRS to then calculate altitude
unsigned long long	m_time	0	64 bit (milliseconds into day value)
char	m_id[16]	0	A..H for addressable, else blank
char	m_sourceName[16]	""	Name label for the Altimeter

Table 21 – AltimeterDataRecord storage Record

3.2.2. **System/Information Records**

These Records contain information that are not specific to any device, although may be related to the functionality of a device. They will contain a number of different Record types, which include:

Marker Inputs for Chart and MicronNav usage
(e.g. can be real time input via port or via user chart addition, else imported through a file)

System Error and Notification events
(e.g. Device Error, Low Disk Space, Lost Comms)

File Bookmarks
(e.g. for Log File Preview purposes – can be added during post file editing or triggered during recording to mark special events)

Note: New record types will be added as the Application develops in the future.

System/Information Records are stored in the log file with the same Common Interface Header as used in storing Sensor/Device Records (*see Section 3.2.1*). The System/Information records are interspersed with Sensor/Device Records in the ".dat" log.

System/Information Records will be stored with the following structure:

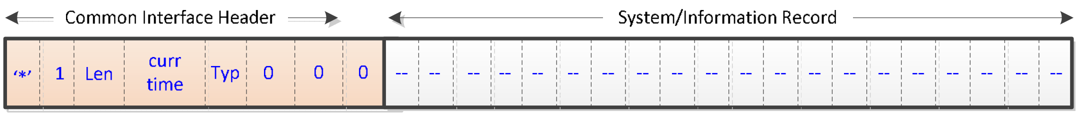

Figure 17 – Example of System/Information Record Structure

Examples of each of the aforementioned supported types now follow:

1. Marker Input

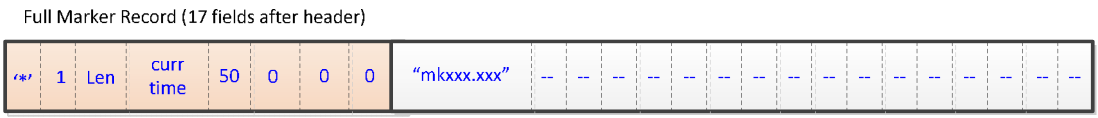

Figure 18 – Example of Full Marker Input Record Structure

The **Full Marker Record (FullMrkRec)** contains all the fields for the drawing and positioning of a marker on the Genesis chart. Its structure, taken from Seanet Pro, is as follows:

Data Type	Variable Name	Default Value	Description
std::string	m_id	"mk0.0"	This is a unique string ID. It comprises a 2 letter header ("mk") followed by a Date/Time code, e.g. "mk42479.5656234375". Any unique string value is acceptable.
unsigned char	m_group	0	Currently unused and set to 0.
double	m_xCoord	0.0	UTM Easting Value.
double	m_yCoord	0.0	UTM Northing value.
double	m_altitude	0.0	UTM Altitude (currently unused)
char	m_utmZonePar	'C'	UTM Zone Parallel (latitudinal letter, e.g. 'C' through 'X').
unsigned char	m_utmZoneMer	1	UTM Zone Meridian (longitudinal number, e.g. 1 through 60).
unsigned char	m_utmEllipsoid	9	UTM Ellipsoid code (0 = Airy, 1 = Australian National, 2 = Bessel1841, 3 = Clarke 1866, 4 = Clarke 1880, 5 = Everest, 6 = GRS80, 7 = International 1924, 8 = Modified Airy, 9 = WGS84).
unsigned int	m_pointSize	0	Point Size. Applies to Circle, Square & Triangle shape types only, otherwise set to 0.
std::string	m_dateTime	""	Date & Time in English (GB) Locale. Format is "dd/mm/yyyy hh:mm:ss".
unsigned char	m_shapeType	0	0 = Circle, 1 = Square, 2 = Triangle, 3 = Sonar Range, 4 = Preset Image (see Image Info below).
unsigned int	m_shapeCol	0	Applies to Circle, Square and Triangle shape types only, otherwise set to 00000000 (32-bit RGBA).
unsigned int	m_fontInnerCol	0	Applies to Comment Text (32-bit RGBA).
unsigned int	m_fontOuterCol	0	Applies to Comment Text (32-bit RGBA).
unsigned char	m_mrkBitwise	7	Bit 1 = Show Marker, Bit 2 = Show Coordinates, Bit 3 = Show Comment Text (i.e. 00000111 = Show All).
std::string	m_imgInfo	""	Presets = Red Flag, Blue Flag, Green Flag, Buoy, Anchor, Rock, Danger, POI, ViewPort, Sonar, Diver, Wheel, Comment or MLO. Alternatively can be full path and name of an image file (e.g. "C:\Image1.bmp").
std::string	m_comment	""	Comment text.

Table 22 – Full Marker Record (FullMrkRec)

(Note: Above is not currently supported)

2. System Event

System Event Record

Figure 19 – Example of System Event Record Structure

(Note: Above is not currently supported)

3. Bookmark

BookMark Record

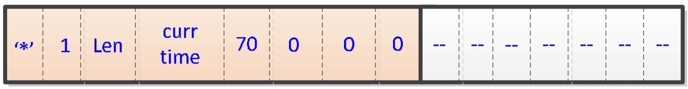

Figure 20 – Example of Bookmark Record Structure

(Note: Above is not currently supported)

SeaKing Bathy records

These records contain the processed data from a SeaKing Bathy. Note: These do not follow the raw data strings that the Bathy sends to the surface computer.

Bathy Record (from Node 40)

Figure 21 – Example of Bathy Record Structure

Where the 'Bathy Record' payload in above structure will contain the processed Bathy data, e.g.

Data Type	**Variable Name**	**Default Value**	**Description**
unsigned char	m_node	40	Node Number of Sensor
unsigned int	m_dqSN	0	Digiquartz Sensor Serial Number
unsigned int	m_ctSN	0	Conductivity/Temperature Probe Serial Number
double	m_depthInMeters	0	Calculated Depth in metres

double	m_altitudeInMeters	0	Altitude in Metres from a connected Altimeter
double	salinity	0	Salinity calculation (ppt)
double	m_conductivity	0	CT Probe Conductivity reading (mS/cm)
double	m_condTemperature	0	CT Probe Temperature reading (deg.C)
double	m_vosLocal	0	Velocity of Sound calculation (metres/sec)
double	m_vosMean	0	Mean Velocity of Sound calculation (e.g. mean over all depths) (metres/sec)
double	m_densityLocal	0	Water Density calculation (g/cm^3)
double	m_densityMean	0	Mean Water Density calculation (e.g. mean over all depths) (g/cm^3)
double	m_seaPressureBar	0	Sea Pressure (Bar)
double	m_dqPressure	0	Raw Digiquartz Pressure, includes Atmospheric Pressure (Bar)
double	m_dqTemperature	0	Digiquartz Temperature (deg.C)
double	m_inTemperature	0	Internal Temperature (deg.C)
double	m_usedBarometer	0	Barometer value applied in Pressure/Depth calculations (millibar)
bool	m_depthValid	false	Depth calculation is valid
bool	m_altimeterValid	false	Altimeter is valid
bool	m_dqValid	false	Digiquartz Sensor readings are valid
bool	m_ctValid	false	CT Sensor readings are valid
bool	m_inTempValid	false	Internal Temperature Sensor reading is valid
bool	m_salinityValid	false	Salinity calculation is valid
bool	m_densityValid	false	Density calculation is valid
bool	m_velocityOSValid	false	Velocity of Sound calculation is valid
unsigned long long	m_time	0	Milliseconds into day (from internal RTC)

Table 23 – BathyDataRecord

4. Appendix – Example Structure of Log File

Note: XML used in this Example

File Start Tag	<genesisLog>
Log File Header	<logHeader> <logStart>4261347071</logStart> . . Etc </logHeader>
Project settings	<GuiCurrentSettings> <application> . . </application> <devices> . . </devices> . . Etc </GuiCurrentSettings>
Device Config Records	<deviceConfigs> <node2> <version>########</version> <userBB>########</userBB> </node2> . . <node100> <licensing>########</licensing> </node100> . . Etc </deviceConfigs>
Log File Terminator	<logTerminator> <logEnd>4294901501</logEnd> . . Etc </logTerminator>
File End Tag	</genesisLog>

5. Appendix – Example calculating range line in meters

Following function calculate the range line in meters in playback mode.

```
float CalculateRangeInMeters(GMainImage& mainImage)
{
   unsigned short rangeCompUsed = 0;
   if (mainImage.m_usRangeCompUsed & 0x20)
   {
      // Absolute compression level
       rangeCompUsed = (mainImage.m_usRangeCompUsed & 0x0F);
   }
   else
   {
      // Encoded compression level
      rangeCompUsed = (1 << (mainImage.m_usRangeCompUsed & 0x000F));
   }

   float RangeLines    = ((float)mainImage.m_uiEndRange / (float)rangeCompUsed);
   float ModulationFreq= ((float) mainImage.m_uiModulationFrequency /
(float)rangeCompUsed );

   float rangeInMeters =  ( RangeLines * (mainImage.m_fSosAtXd / 2.0) / ModulationFreq);
   std::cout<< " Range in Meters " << rangeInMeters << std::endl;
   return rangeInMeters;
}
```