

GSFlib, The Generic Sensor Format Library

24 Sep, 2010

Prepared For:
Naval Oceanographic Office
Stennis Space Center, MS 39522

Prepared By:
Science Applications International Corporation
221 Third Street
Newport, RI 02840

GSFLib, the Generic Sensor Format Library

REVISIONS			
Rev	Date	Pages Affected	Remarks
0	04 SEP 1998	All	Baseline Version
1	12 NOV 1998	All	Updated specification to reflect changes due to implementations through GSF-v1.07.
2	07 OCT 1999	All	Updated specification to reflect changes due to implementations through GSF-v1.08.
3	12 OCT 1999	All	Updated specification to reflect changes due to implementations through GSF-v1.09.
4	20 OCT 2000	All	Updated specification to reflect changes due to implementations through GSF-v1.10
5	16 JAN 2001	All	Updated specification to reflect changes due to implementations through GSF-v1.11
6	29 MAR 2002	Various	Updated Library Documentation to reflect changes made for GSF version 2.0. Including: (c++ support, and support for Simrad EM120)
7	08 JUL 2002	Various	Updated Library Documentation to reflect changes made for GSF version 2.01.
8	20 JUN 2003	Various	Updated Library Documentation to reflect changes made for GSF version 2.02, including support for bathymetric receive beam time series intensities.
9	29 DEC 2004	Various	Updated Library Documentation to reflect changes made for GSF version 2.03.
10	30 JUN 2006	Various	Updated Library Documentation to reflect changes made for GSF version 2.04.
11	09 MAR 2007	Various	Updated Library Documentation to reflect changes made for GSF version 2.05.
12	04 Sep 2007	Various	Updated Library Documentation to reflect changes made for GSF version 2.06 and 2.07
13	03 Dec 2007	None	No change to the text for GSF version 2.08
14	30 Jan 2008	Various	Updated Library Documentation to reflect changes made for GSF version 2.09
15	20 Mar 2009	Various	Updated Library Documentation to reflect changes made for GSF version 03.01
16	24 Sep 2010	Various	Updates for GSF version 03.02.

Table of Contents

1.Introduction	1-1
1.1Implementation Concept	1-1
1.2Development History	1-3
1.3Restrictions and Limitations	1-6
1.4References	1-7
1.5Distribution	1-7
1.6Sensors Supported.....	1-7
1.7Computer Platforms Supported.....	1-8
1.8Documentation Conventions	1-8
2.Function Definitions	2-1
2.1Access Functions.....	2-1
2.1.1Function: gsfOpen	2-1
2.1.2Function: gsfOpenBuffered.....	2-2
2.1.3Function: gsfRead	2-4
2.1.4Function: gsfWrite.....	2-5
2.1.5Function: gsfSeek.....	2-6
2.1.6Function: gsfClose.....	2-7
2.2Utility Functions	2-8
2.2.1Function: gsfCopyRecords	2-8
2.2.2Function: gsfFree.....	2-8
2.2.3Function: gsfPutMBParams	2-9
2.2.4Function: gsfGetMBParams	2-9
2.2.5Function: gsfLoadScaleFactors	2-10
2.2.6Function: gsfGetScaleFactors	2-12
2.2.7Function: gsfSetDefaultScaleFactor.....	2-13
2.2.8Function: gsfLoadDepthScaleFactorAutoOffset.....	2-14
2.2.9Macro: gsfTestPingStatus	2-15
2.2.10Macro: gsfSetPingStatus	2-15
2.2.11Macro: gsfClearPingStatus.....	2-16
2.3Information Functions.....	2-16
2.3.1Function: gsfPrintError	2-16
2.3.2Function: gsfStringError	2-17
2.3.3Function: gsfIndexTime	2-17
2.3.4Function: gsfPercent.....	2-18
2.3.5Function: gsfGetNumberRecords.....	2-18
2.3.6Function: gsfGetSwathBathyBeamWidths	2-19
2.3.7Function: gsfGetSwathBathyArrayMinMax	2-20
2.3.8Function: gsfIsStarboardPing.....	2-20
2.3.9Function: gsf_register_progress_callback.....	2-21
2.3.10Function: gsfGetSonarTextName.....	2-21
2.3.11Function: gsfFileSupportsRecalculateXYZ	2-22
2.3.12Function: gsfFileSupportsRecalculateTPU	2-23
2.3.13Function: gsfFileSupportsRecalculateNominalDepth.....	2-23
2.3.14Function: gsfFileContainsMBAmplitude.....	2-24

2.3.15Function: gsfFileContainsMBImagery.....	2-25
2.3.16Function: gsflsNewSurveyLine.....	2-26
3.ERROR CODE DESCRIPTIONS	3-28
4.C-language Definitions of Structures used by GSFLib.....	4-1
4.1Definition of GSF Data Records	4-1
4.1.1Header Record	4-1
4.1.2Swath Bathymetry Ping Record	4-1
4.1.3Single-beam Bathymetry Record	4-20
4.1.4Sound Velocity Profile (SVP) Record	4-21
4.1.5Processing Parameters Record	4-22
4.1.6Sensor Parameters Record.....	4-23
4.1.7Comment Record.....	4-23
4.1.8History Record	4-24
4.1.9Navigation Error Record	4-24
4.1.10Swath Bathymetry Summary Record.....	4-24
4.1.11Attitude Record	4-25
4.2Supporting Data Structures and Definitions	4-25
4.2.1Record Identifier	4-25
4.2.2Time Structure.....	4-25
4.2.3Null values used to represent missing data	4-25
4.2.4Positioning System Type Codes.....	4-26

List of Figures

Figure 1-1 GSFLib Functions	1-2
-----------------------------------	-----

List of Tables

Table 2-1 GSF Beam Array Field Size Definitions	2-11
Table 3-1 GSF Error Codes	3-28
Table 4-1 Sensor ID allocation to Sensor Specific Subrecord Data Structure	4-17

1. INTRODUCTION

The Generic Sensor Format (GSF) library contains functions for creating and accessing multibeam sonar data that have been stored in a generic byte stream format corresponding to the sequential encapsulation described in the Generic Sensor Format Specification. This specification defines a set of ten record types that are used to store bathymetric data. This document describes the library that supports GSF format version 03.01.

This document is derived from documentation within the GSFLib source code, primarily the header file, `gsf.h`. The intent is to present that information in a more accessible, organized form and to describe the library's design and implementation. Because the information presented herein is derived from the source code, the code itself should be the primary reference for application developers.

1.1 Implementation Concept

The GSF library (gsflib) is a “thin” layer of software that transfers data between the data format described in the specification and a standardized set of data structures. This is necessary because the specified data format is a byte stream of data containing records of arbitrary length that have been extensively optimized for compactness and is not easily manipulated. The organization of the data structures populated by GSFLib is for the developer's convenience and presents the data in a uniform manner with a consistent set of physical units. There is a one-to-one correspondence between the record types defined in the specification and the data structures made available through the library.

Figure 1-1 illustrates the GSF library functions. There are three functional categories in the library routines: those that provide access to the data when stored on disk, those that perform utility operations and those that provide information about the data. The access functions, which translate between the memory-based data structures and the byte-stream data format, include operations to open and close, read and write to data files and seek functions to access data by time and record type.

Utility functions include routines that copy data structures, free memory, translate processing parameters into a more accessible form, and provide the programmer with access to the scale factors used to optimize the storage to ping arrays. Processing parameters document the extent to which data have been processed and the values of any correctors or offsets that have been applied to the data. Access to processing parameters is necessary when they are required or need to be updated. Scale factor information defines how the data are packaged into the GSF data files. They are automatically applied to read operations and need to be manipulated only when the application is writing data to disk.

Informational functions provide a variety of facts about the data. These functions provide capabilities such as:

- describing error conditions,
- returning the relative location of the file pointer within the file,
- providing counts of the number of records of a given type,

- discriminating between starboard and port-directed beams in dual transducer configurations and
- Providing beam widths for the data being processed.
- Providing the name of the sensor

It should be noted that for some sonars this beam width information is not stored within the data but is provided by lookup tables within the library source code.

The GSF byte stream is a sequentially oriented file but the library provides for direct access to the data via an auxiliary index file. Upon opening a data file for direct access, the disk is inspected for an index file that corresponds to the data file being opened. If there is no index file, one is created. The index file provides direct access to any record in the data file. The creation and maintenance of the index file is transparent to both the application developer and to the user. The normal sequence of events is for the data file to be written sequentially and for the index file to be created by the first program that needs to examine it using direct access. At this time, the index file format is not a part of the GSF data specification but is defined only within the library.

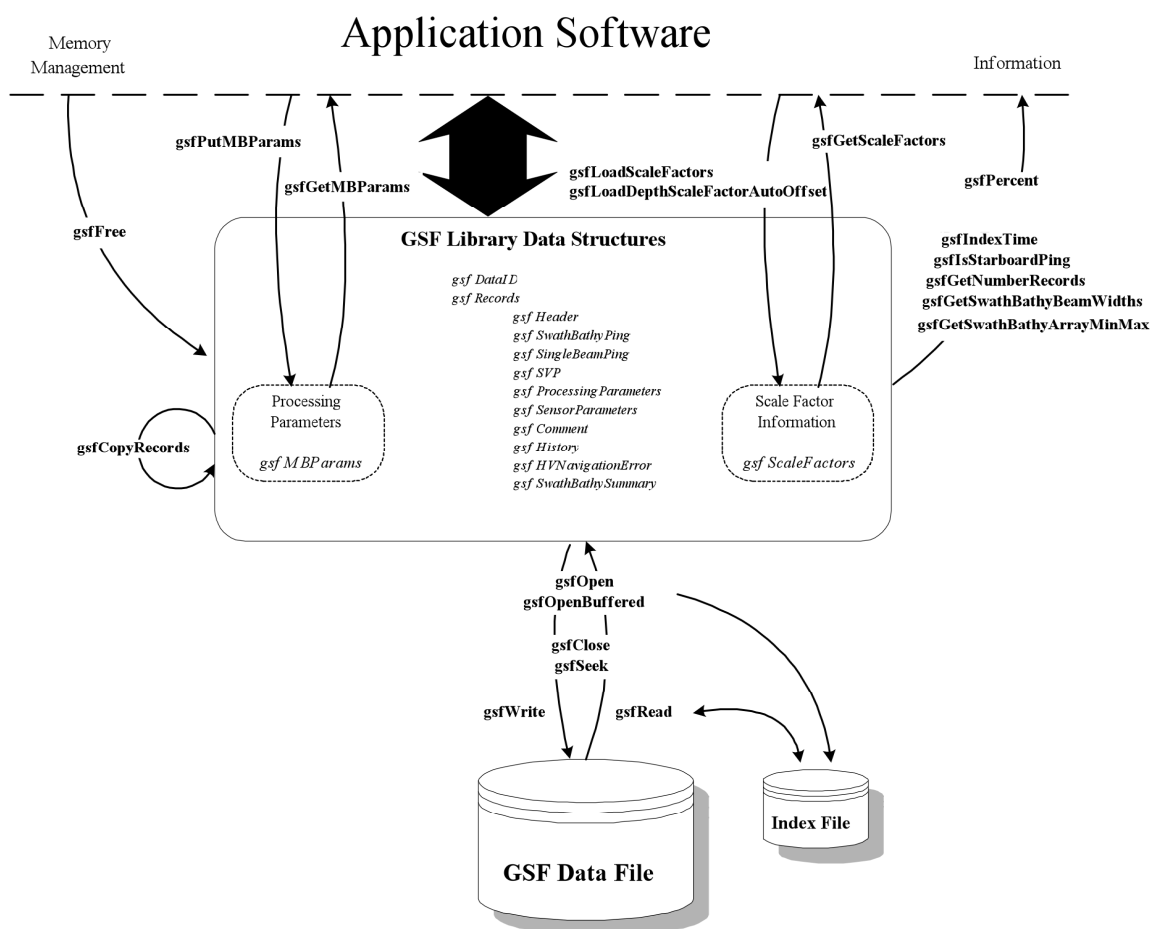


Figure 1-1 GSFLib Functions

1.2 Development History

J. Scott Ferguson and Brad Ward of SAIC and Daniel Chayes of the Naval Research Lab developed the GSF specification. The Defense Mapping Agency supported its development and it was first published on 31 March 1994. The primary author of the GSF library is John Shannon Byrne of SAIC and was first released on 3 May 1994. The U.S. Naval Oceanographic Office (NAVOCEANO) and Naval Sea Systems Command (NAVSEA) supported the development of this library. NAVOCEANO also provided significant direction and feedback during the library's development and initial deployment. After deployment, the GSF Working Group was formed. This group discusses issues relative to the specification and the library, provides direction for GSF development and acts as a configuration control board to accept updates. The working group exchanges technical information mostly via email. As of March 2007, the GSF mailing list (gsf@navo.navy.mil) is no longer available. The new GSF mailing list can be subscribed to by filling out the form located here: <http://www.saic.com/maritime/gsf/form.asp>. Both the specification and the GSF library are maintained under configuration control by NAVOCEANO.

The library's release history is as follows:

Release Date	Version ID	Description
03 May 1994	GSF-v01.00	Initial Release.
14 Aug 1995	GSF-v01.01	Direct and sequential access now works through common gsfRead and gsfWrite API. All pointers to dynamically allocated memory are now maintained by the library.
22 Dec 1995	GSF-v01.02	Added gsfGetMBParams , gsfPutMBParams , gsfIsStarboardPing , and gsfGetSwathBathyBeamWidths . Also added <i>GSF_APPEND</i> as a file access mode, and modified <i>GSF_CREATE</i> access mode so that files can be updated (read and written).
20 Aug 1996	GSF-v01.03	Added support for single beam echosounders. Added gsfStringError function.
24 Mar 1997	GSF-v01.04	Added support for RESON 8101 sonar and enhanced support for "classic" Seabeam sonar. Increased the maximum record size from 4 kbytes to 32 kbytes.
04 Sep 1998	GSF-v01.06	Added support for SeaBeam 2100 series multibeam sonars and for Elac Bottomchart MkII sonars. Minor enhancements to code portability.
12 Nov 1998	GSF-v01.07	Defined a new GSF navigation error record <i>gsfHVNavigationError</i> that replaces the currently defined navigation error record <i>gsfNavigationError</i> . Modified encode of the existing error array subrecords (depth_error, across_track_error, and along_track_error) as two byte quantities. Added two new array subrecords to the GSF swath bathymetry ping data structure, namely horizontal error and vertical error. Modified the gsfPrintError function so that it calls the gsfStringError function.

		gsfStringError function expanded so that all defined error conditions are handled.
07 Oct 1999	GSF-v01.08	Added support for Simrad multibeam models EM-3000, EM-1002 and EM-300, as well as added a new compressed SASS (<i>gsfCmpSassSpecific</i>) specific data structure. Added two new functions gsfGetSwathBathyArrayMinMax and gsfLoadDepthScaleFactorAutoOffset in support of signed depth. Also added processing in the gsfGetSwathBathyBeamWidths function to return the beam width values specified within the EM-3000 series data formats. Increased the <i>GSF_MAX_PROCESSING_PARAMETERS</i> macro from sixty-four to one hundred and twenty-eight and the <i>GSF_MAX_SENSOR_PARAMETERS</i> macro from thirty-two to one hundred and twenty-eight. Modified gsfPutMBParameters function to allow processing parameters to contain the appropriate designator for the vertical datum.
12 Oct 1999	GSF-v01.09	Updated the contents of the compressed SASS (<i>gsfCmpSassSpecific</i>) specific subrecord. Added a comment block to the compressed SASS specific subrecord definition to describe the mapping between SASS and GSF data. Included annotations informing that the <i>gsfCmpSassSpecific</i> data structure is intended to replace the <i>gsfTypeIIISpecific</i> data structure in a future release. All new coding should use the <i>gsfCmpSassSpecific</i> data structure.
20 Oct 2000	GSF-v01.10	Enhancements for index file portability between big and little endian-based host machines. Updates to source code for minor bug fixes.
16 Jan 2001	GSF-v01.11	Updated the contents of the <i>gsfEM3RunTime</i> data structure to include separate elements for port and starboard swath width and for port and starboard coverage sectors. Updated the contents of the <i>gsfEM3RunTime</i> data structure to include the HiLo frequency absorption coefficient ratio. Added checks for LINUX specific defines before defining timespec structure. Added support for more tidal datums. Fixed errors in decoding of HV Navigation Error records.
29 Mar 2002	GSF-v02.00	Modified to support access from c++ applications, address file sharing problems on multiprocessor Linux configurations, resolve compile macros used for Win32, resolved several minor bug fixes, remove unused automatic variables, add support for the Simrad EM120 sonar, reserve subrecord IDs for the latest datagram format for Reson 8101, 8111, 8125, 8150, and 8160 sonar systems, and ensure that a string terminating NULL is applied when strncpy is used.

08 Jul 2002	GSF-v02.01	Added gsfAttitude record to allow storage of full time series of attitude data. Added a new sensor specific subrecord for Reson 8101, 8111, 8125, 8150, and 8160 sonar systems. Expanded the gsfMBOffsets structure to include motion sensor offsets. Updated gsfGetMBParams and gsfPutMBParams to encode and decode new motion sensor offsets in the process_parameters record.
20 Jun 2003	GSF-v02.02	Added support for bathymetric receive beam time series intensity data. Added sensor-specific single-beam information to the multibeam sensor specific subrecords.
29 Dec 2004	GSF-v02.03	Fixed memory leaks, fixed encoding and decoding of 1-byte BRB intensity values, updated gsfLoadDepthScaleFactorAutoOffset to vary the offset interval based on precision, added beam spacing to Reson 8100 sensor-specific subrecord, reserved sensor Ids for Simrad EM3002, EM3002D, and EM3000D, added sensor specific support for Reson Navisound singlebeam, added copy of vertical_error and horizontal_error arrays in gsfCopyRecords, and added definitions for RTG position type to gsfHVNavigationalError record.
30 Jun 2006	GSF-v2.04	Added support for EM121A data received via Kongsberg SIS. Added support for EM3000D and EM3002D in gsfIsStarboard ping function. Added new service to allow calling programs to register a callback function for reporting progress of index file creation. Updated gsfCopyRecords to copy all HV Nav Error data from source to target data structure. Updates to support compilation on 64-bit architectures, and compilation on MAC OSX operating system.
09 Mar 2007	GSF-v2.05	Added support for bathymetry data from the GeoAcoustics Ltd. GS+ Interferometric side-scan sonar system. Reserve sub-record IDs for the Kongsberg EM122, EM302, and EM710 systems.
04 Sep 2007	GSF-v2.06, GSF-v2.07	Added support for the Kongsberg EM122, EM302, and EM710 multibeam systems. Added application level control over the field size to be used for a subset of the beam array subrecords. Improved error checking in gsfLoadScaleFactor(). Fixed a problem in DecodeSignedByteArray that was only an issue on the SGI platform.
03 Dec 2007	GSF-v2.08	Modified the approach used to parse the beam array subrecords to no longer depend on the compression flag field of the scale factor subrecord for determining the field size. This dependency on the compression flag field was added in GSFv2.06 on the premise that a default value of zero could (always) be expected.

30 Jan 2008	GSF-v2.09	Added support for Klein 5410 Bathymetric Sidescan.
20 Mar 2009	GSF-v03.01	Added support for the Reson 7125 and EM2000. Added fields for height, separation, and gps tide corrector to the <code>gsfSwathBathyPing</code> record. Added new processing parameter record values: <code>vessel_type</code> , <code>full_raw_data</code> , <code>msb_applied_to_attitude</code> , <code>heave_removed_from_gps_tc</code> . Added new sensor ids for EM3 sensors to differentiate between data logged from the depth datagram and the raw range and beam angle datagram.
24 Sep 2010	GSF-v03.02	Added support for KM2040. Added support for Imagenex Delta-T. Add new query functions to provide calling applications with a simple means to determine what data are contained in the GSF file and what processing operations can be supported given the parameters available in the input file. Added separation uncertainty field to the Navigation uncertainty record. Several bugs resolved.

1.3 Restrictions and Limitations

The following restrictions or limitations apply to the GSFLib code.

- The library assumes the host computer uses the ASCII character set.
- The library is written in the C language and assumes that the type `short` is 16 bits, and that the type `int` is 32 bits.
- The library provides access to individual data files only and does not support the development of metadata or transmittal files. It should be noted, however, that many of the data items recorded in the files' summary and parameter records may be used to populate metadata records.
- Data compression flags are maintained within the ping scale factors subrecord but data compression is not supported.
- The index function creates separate index files that make assumptions about the file naming convention. The library names the index file the same as the data file name but replaces the third to the last character with an "n". This is because the files are expected to be named using a file naming convention adhered to within NAVOCEANO for data collected by their Integrated Survey Systems (ISS and ISS-60). No protection exists for the case where a GSF data file already has an "n" in the third to the last character.
- Time is recorded in precise form only with fractional seconds included in all time fields. The beginning of the epoch is required to be midnight of 1 January 1970, thus data recorded prior to this date is not supported.
- The only horizontal datum supported is "WGS-84"; supported tidal datums include "UNKNOWN", "MLLW", "MLW", "ALAT", "ESLW", "ISLW", "LAT", "LLW", "LNLW", "LWD", "MLHW", "MLLWS", "MLWN", and "MSL". This is a limitation with the data structure `gsfMBParams` which represents horizontal and vertical datums as integers. Only these datums have integer definitions in `gsf.h`.
- Data record compression is not supported.
- The current version of GSFLib library does provide text string translations for all error code returns; however, all definitions do not have unique values.

- The name of the *gsfSwathBathySummary* record implies that the data in this structure is specific to the Swath Bathy Ping Record. This is not the case; the data structure is implemented to represent the Summary Record as defined in the specification.
- The index file is not portable between 32-bit and 64-bit computers.

1.4 References

Generic Sensor Format Specification, 16 January 2001, Prepared for: Naval Oceanographic Office, Stennis Space Center, MS, by Science Applications International Corporation, 221 Third Street, Newport RI.

1.5 Distribution

The information in this document and the GSF library source code itself is unclassified and may be distributed without restriction.

1.6 Sensors Supported

Multibeam echosounders

- Elac Bottomchart Mk II
- RESON SEABAT 9000 Series
- RESON 7125
- RESON 8101
- RESON 8111
- RESON 8125
- RESON 8150
- RESON 8160
- SeaBeam 2100 series
- Simrad EM100
- Simrad EM121
- Simrad EM300
- Simrad EM950
- Simrad EM1000
- Simrad EM1002
- Simrad EM2000
- Simrad EM3000 and EM3000D
- Simrad EM120
- Simrad EM3002 and EM3002D
- Simrad EM122
- Simrad EM302
- Simrad EM710
- Simrad EM2040
- Imagenex Delta-T

Interferometric Side-Scan Systems

- SEAMAP
- GeoAcoustics GS+

Multibeam Archival Formats

- Compressed SASS
- Single-beam Echosounders
- Odom Echotrac
 - ODEC Bathy2000
 - Reson Navisound
- Single-beam Archival Formats
- MGD77
 - BDB
 - NOS HDB
- Bathymetric Sidescan Systems
- Klein 5410

1.7 Computer Platforms Supported

The GSF library has been used on the following platforms:

- HP Series 7000 workstations running HPUX 9.0, 10.0, or 10.20, or 11.0
- PCs running IBM OS/2, versions 2.0, 3.0 and 4.0, LINUX (32 bit and 64 bit), and WINDOWS NT/2000/XP
- Digital Alpha Workstation running Digital UNIX, version ***
- Silicon Graphics running IRIX 6.3
- Sun ***
- Mac OSX

1.8 Documentation Conventions

- References to GSF functions are **bolded**.
- References to GSF data structures or definitions are *italicized*.
- Function prototypes, function arguments and other references to C-language source code are in Courier type (e.g., `int`)

2. FUNCTION DEFINITIONS

The library function definitions in this section are in three functional categories, those used to access data, those used to perform utility functions, and those that provide information about the data.

2.1 Access Functions

Access functions include those used to open and close data files, read and write data and place the file pointer as various locations within the file.

2.1.1 Function: **gsfOpen**

Usage:

```
int gsfOpen(const char *filename,
            const int   mode     ,
            int         *handle  )
```

Description:

This function attempts to open a GSF data file. If the file exists and is opened for read-only or for update, the GSF header is read to confirm that this is a GSF data file. If the file is opened for creation, the GSF header containing the version number of the software library is written into the header. This function passes an integer handle back to the calling application. The handle is used for all further access to the file. **gsfOpen** explicitly sets stream buffering to the value specified by *GSF_STREAM_BUF_SIZE*. The internal file table is searched for an available entry whose name matches that specified in the argument list, if no match is found, then the first available entry is used. Up to *GSF_MAX_OPEN_FILES* files may be open by an application at a time.

If a file is opened as *GSF_READONLY_INDEX* or *GSF_UPDATE_INDEX* a corresponding index file is expected to exist. If the index file exists, its contents are examined to determine if the GSF file has increased in size since the index file was created. If not, subsequent file accesses use the index file. If the index file does not exist, the **gsfOpen** function automatically creates it. If the GSF file is larger than that recorded in the index file, the index file is updated to correspond to the new records in the GSF file.

Inputs:

filename	a fully qualified path to the GSF file to be opened
mode	may have the following values: <ul style="list-style-type: none"><i>GSF_READONLY</i> open an existing file for read-only access<i>GSF_UPDATE</i> open an existing file for reading and writing<i>GSF_CREATE</i> create a new GSF file<i>GSF_READONLY_INDEX</i> open an existing file for read only access with an index file<i>GSF_UPDATE_INDEX</i> open an existing file for reading an writing with an index file

handle a pointer to an integer to be assigned a handle which will be referenced for all future file access.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_ACCESS_MODE
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR
GSF_FOPEN_ERROR
GSF_READ_ERROR
GSF_SETVBUF_ERROR
GSF_TOO_MANY_OPEN_FILES
GSF_UNRECOGNIZED_FILE
GSF_OPEN_TEMP_FILE_FAILED

2.1.2 Function: gsfOpenBuffered

Usage:

```
int gsfOpenBuffered(const char *filename,
                    const int  mode,
                    int        *handle,
                    int         buf_size)
```

Description:

This function attempts to open a GSF data file. If the file exists and is opened read-only or for update, the GSF header is read to confirm that this is a GSF data file. If the file is opened for creation, the GSF header containing the version number of the software library is written into the header. This function passes an integer handle back to the calling application. The handle is used for all further access to the file. **gsfOpenBuffered** explicitly sets stream buffering to the value specified by the *buf_size* argument. The internal file table is searched for an available entry whose name matches that specified in the argument list, if no match is found, then the first available entry is used. Up to *GSF_MAX_OPEN_FILES* files may be open by an application at a time. **gsfOpenBuffered** performs identical processing to **gsfOpen** except that the caller is allowed to explicitly set the I/O buffer size.

If a file is opened as *GSF_READONLY_INDEX* or *GSF_UPDATE_INDEX*, a corresponding index file is expected to exist. If the index file exists, its contents are examined to determine if the GSF file has increased in size since the index file was created. If not, the index file is used for subsequent file accesses. If the index file does not exist, the **gsfOpenBuffered** function automatically creates it. If the GSF file is larger than that recorded in the index file, the index file is updated to correspond to the new records in the GSF file.

Inputs:

filename a fully qualified path to the GSF file to be opened

mode may have the following values:

- GSF_READONLY* open an existing file for read-only access
- GSF_UPDATE* open an existing file for reading and writing
- GSF_CREATE* create a new GSF file
- GSF_READONLY_INDEX* open an existing file for read-only access with an index file
- GSF_UPDATE_INDEX* open an existing file for reading and writing with an index file

handle a pointer to an integer to be assigned a handle which will be reference for all future file access.

buf_size an integer buffer size in bytes.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

- GSF_BAD_ACCESS_MODE*
- GSF_FILE_SEEK_ERROR*
- GSF_FLUSH_ERROR*
- GSF_FOPEN_ERROR*
- GSF_READ_ERROR*
- GSF_SETVBUF_ERROR*
- GSF_TOO_MANY_OPEN_FILES*
- GSF_UNRECOGNIZED_FILE*
- GSF_OPEN_TEMP_FILE_FAILED*

2.1.3 Function: **gsfRead**

Usage:

```
int gsfRead(int          handle,
            int          desiredRecord,
            gsfDataID    *dataID,
            gsfRecords    *rptr,
            unsigned char *buf,
            int           max_size)
```

Description:

gsfRead supports both direct and sequential access. If the file is opened for sequential access, this function reads the desired record from the GSF data file specified by the handle. Setting the `desiredRecord` argument to *GSF_NEXT_RECORD* reads the next record in the data file. The `desiredRecord` record may be set to specify the record of interest. In this case, the file is read, skipping past intervening records. After locating the desired record, it is read and decoded from external to internal form. If the data contains the optional checksum, the checksum is verified. All of the fields of the *gsfDataID* structure, with the exception of the `record_number` field will be loaded with the values contained in the GSF record byte stream. For sequential access, the `record_number` field is undefined. The `buf` and `max_size` arguments are normally set to NULL, unless the calling application requires a copy of the GSF byte stream.

If the file is opened for direct access, then the combination of the `recordID` and the `record_number` fields of the *dataID* structure are used to uniquely identify the record of interest. The address for this record is retrieved from the index file, which was created on a previous call to **gsfOpen** or **gsfOpenBuffered**. If the record of interest is a ping record that needs new scale factors, the ping record containing the scale factors needed is read first, and then the ping record of interest is read. Direct access applications must set the `desiredRecord` argument equal to the `recordID` field in the *gsfDataID* structure.

Inputs:

<code>handle</code>	the handle to the file as provided by gsfOpen
<code>desiredRecord</code>	the desired record or <i>GSF_NEXT_RECORD</i>
<code>dataID</code>	a pointer to a <i>gsfDataID</i> structure to be populated for the input record.
<code>rptr</code>	a pointer to a <i>gsfRecords</i> structure to be populated with the data from the input record in internal form.
<code>buf</code>	an optional pointer to caller memory to be populated with a copy of the GSF byte stream for this record.
<code>max_size</code>	an optional maximum size to copy into <code>buf</code>

Returns:

This function returns the number of bytes read if successful or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_ATTITUDE_RECORD_DECODE_FAILED
GSF_BAD_FILE_HANDLE
GSF_CHECKSUM_FAILURE
GSF_COMMENT_RECORD_DECODE_FAILED
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR
GSF_HEADER_RECORD_DECODE_FAILED
GSF_HISTORY_RECORD_DECODE_FAILED
GSF_HV_NAV_ERROR_RECORD_DECODE_FAILED
GSF_INSUFFICIENT_SIZE
GSF_NAV_ERROR_RECORD_DECODE_FAILED
GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
GSF_READ_ERROR
GSF_READ_TO_END_OF_FILE
GSF_RECORD_SIZE_ERROR
GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
GSF_SUMMARY_RECORD_DECODE_FAILED
GSF_SVP_RECORD_DECODE_FAILED
GSF_UNRECOGNIZED_RECORD_ID
GSF_UNRECOGNIZED_SUBRECORD_ID

2.1.4 Function: **gsfWrite**

Usage:

```

int gsfWrite(int      handle,
             gsfDataID *id,
             gsfRecords *rptr)

```

Description:

gsfWrite encodes the data from internal to external form, and then writes the requested record into the file specified by handle, where handle is the value returned by **gsfOpen**. The record is written to the current file pointer for handle. An optional checksum may be computed and encoded with the data if the checksum flag is set in the *gsfDataID* structure. If the file is opened for sequential access (*GSF_CREATE*, or *GSF_UPDATE*) then the `recordID` field of the *gsfDataID* structure is used to specify the record to be written.

When opening the file for direct access (*GSF_UPDATE_INDEX*), the combination of the `recordID` and the `record_number` fields of the *gsfDataID* structure uniquely identify the record to write. The address of the record of interest is read from the index file and the file pointer is moved to this offset before the record is encoded and written to disk.

Inputs:

handle	the handle for this file as returned by gsfOpen
id	a pointer to a <i>gsfDataID</i> containing the record ID information for the record to write.
rptr	a pointer to a <i>gsfRecords</i> structure from which to get the internal form of the record to be written to the file.

Returns:

This function returns the number of bytes written if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_ATTITUDE_RECORD_ENCODE_FAILED
GSF_BAD_FILE_HANDLE
GSF_COMMENT_RECORD_ENCODE_FAILED
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR
GSF_HEADER_RECORD_ENCODE_FAILED
GSF_HISTORY_RECORD_ENCODE_FAILED
GSF_HV_NAV_ERROR_RECORD_ENCODE_FAILED
GSF_NAV_ERROR_RECORD_ENCODE_FAILED
GSF_PROCESS_PARAM_RECORD_ENCODE_FAILED
GSF_SENSOR_PARAM_RECORD_ENCODE_FAILED
GSF_SINGLE_BEAM_ENCODE_FAILED
GSF_SUMMARY_RECORD_ENCODE_FAILED
GSF_SVP_RECORD_ENCODE_FAILED
GSF_UNRECOGNIZED_RECORD_ID
GSF_UNRECOGNIZED_SENSOR_ID
GSF_WRITE_ERROR

2.1.5 Function: gsfSeek

Usage:

```
int gsfSeek(int handle,  
            int option)
```

Description:

This function moves the file pointer for a previously opened GSF file.

Inputs:

handle the integer handle returned from **gsfOpen**
option the desired action for moving the file pointer, where:
 GSF_REWIND moves the pointer to first record in the file.
 GSF_END_OF_FILE moves the pointer to the end of the file.
 GSF_PREVIOUS_RECORD backup to the beginning of the record just written or just read.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_BAD_SEEK_OPTION
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR

2.1.6 Function: **gsfClose**

Usage:

```
int gsfClose(const int handle)
```

Description:

This function closes a GSF file previously opened using **gsfOpen**.

Inputs:

handle the handle of the GSF file to be closed.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_FILE_CLOSE_ERROR

2.2 Utility Functions

Utility functions include those used to copy records, to free memory and to access multibeam processing parameters and scale factors.

2.2.1 Function: **gsfCopyRecords**

Usage:

```
int gsfCopyRecords (gsfRecords *target,  
                    gsfRecords *source)
```

Description:

This function copies all of the data contained in the source *gsfRecords* data structure to the target *gsfRecords* data structure. The target *must* be memset to zero before the first call to **gsfCopyRecords**. This function allocates dynamic memory that is NOT maintained by the library. The calling application must release the memory allocated by maintaining the target data structure as static data, or by using **gsfFree** to release the memory.

Inputs:

target	a pointer to a <i>gsfRecords</i> data structure allocated by the calling application, into which the source data is to be copied.
source	a pointer to a <i>gsfRecords</i> data structure allocated by the calling application, from which data is to be copied.

Returns:

This function returns zero if successful, or -1 if an error occurs. *gsfError* is set to indicate the error.

Error Conditions:

GSF_MEMORY_ALLOCATION_FAILED

2.2.2 Function: **gsfFree**

Usage:

```
void gsfFree (gsfRecords *rec)
```

Description:

This function frees all dynamically allocated memory from a *gsfRecords* data structure, and then clears all the data elements in the structure.

Inputs:

rec	pointer to a <i>gsfRecords</i> data structure
-----	---

Returns:

None

Error Conditions:

None

2.2.3 Function: **gsfPutMBParams**

Usage:

```
int gsfPutMBParams (gsfMBParams *p,  
                   gsfRecords *rec,  
                   int handle,  
                   int numArrays)
```

Description:

This function moves swath bathymetry sonar processing parameters from internal form to "KEYWORD=VALUE" form. The internal form parameters are read from an *gsfMBParams* data structure maintained by the caller. The "KEYWORD=VALUE" form parameters are written into the *gsfProcessingParameters* structure of the *gsfRecords* data structure maintained by the caller. Parameters for up to two pairs of transducers are supported.

Inputs:

<i>p</i>	a pointer to the <i>gsfMBParams</i> data structure which contains the parameters in internal form.
<i>rec</i>	a pointer to the <i>gsfRecords</i> data structure into which the parameters are to be written in the "KEYWORD=VALUE" form.
<i>handle</i>	the integer handle to the file set by gsfOpen .
<i>numArrays</i>	the integer value specifying the number of pairs of arrays that need to have separate parameters tracked.

Returns:

This function returns zero if successful, or -1 if an error occurs. *gsfError* is set to indicate the error.

Error Conditions:

None.

2.2.4 Function: **gsfGetMBParams**

Usage:

```
int gsfGetMBParams (gsfRecords *rec,  
                   gsfMBParams *p,  
                   int *numArrays)
```

Description:

This function moves swath bathymetry sonar processing parameters from external form to internal form. The external "KEYWORD=VALUE" format parameters are read from a *gsfProcessingParameters* structure of the *gsfRecords* data structure maintained by the caller.

The internal form parameters are written into a *gsfMBParams* data structure maintained by the caller. Parameters for up to two pairs of transducers are supported.

Inputs:

<i>rec</i>	a pointer to the <i>gsfRecords</i> data structure from which the parameters in "KEYWORD=VALUE" form are to be read.
<i>p</i>	a pointer to the <i>gsfMBParams</i> data structure which will be populated.
<i>numArrays</i>	the integer value specifying the number of pairs of arrays which need to have separate parameters tracked.

Returns:

This function returns zero if successful, or -1 if an error occurs. *gsfError* is set to indicate the error.

Error Conditions:

None.

2.2.5 Function: **gsfLoadScaleFactors**

Usage:

```
int gsfLoadScaleFactor(gsfScaleFactors *sf,
                      int             subrecordID,
                      char             c_flag,
                      double           precision,
                      int              offset)
```

Description:

gsfLoadScaleFactors is used to load the swath bathymetry ping record scale factor structure. This function allows the calling application to specify the precision and offset values used to scale the data from internal form (engineering units) to external form (scaled integer). This function need only be used by applications that are creating a new GSF file from some other data format, or by applications that are updating the numerical values of the beam arrays. In these cases, the application program needs to be aware of the desired data resolution for each beam array and the available dynamic range for each beam array. This is necessary to achieve the desired resolution while avoiding an overflow of the scaled dynamic range. The library does not monitor the scaled values for field level overflow, and no error value will be returned if an overflow occurs. This function should be called at least once for each beam array data type contained in your data, and must be called prior to calling **gsfWrite** by applications creating a new GSF file.

gsfLoadScaleFactors can be called for each beam array before each call to **gsfWrite** to achieve the proper field resolution for each ping record. **gsfLoadScaleFactors** populates the *gsfScaleFactors* sub-structure contained within the *gsfRecords* structure. **gsfWrite** will encode the optional *gsfScaleFactors* sub-record once at the beginning of the data file and again whenever the scale factor values change. Once written, the offset and precision for each beam array remain

in effect for subsequent data records until the scale factors are changed. On encode from internal form to external form, each beam array value is scaled by adding the specified offset and multiplying by one over the specified precision, or:

$$scaled_value = (beam_value + offset) / precision$$

On decode from external form to internal form, the inverse operation is performed, or:

$$beam_value = (scaled_value \times precision) - offset$$

Table 2-1 describes the storage available for each of the array values, and shows the dynamic range of the external form value after the offset and multiplier scaling values are applied. It should be noted that some of the beam arrays support more than one option for the field size. When first creating a GSF file, the calling application can specify the desired field size via the `c_flag` argument to the **gsfLoadScaleFactor** function. The default field size values for each beam array are listed in the table below. The field size is set by using one of the field size macros defined in `gsf.h`. Supported values include: `GSF_FIELD_SIZE_DEFAULT`, `GSF_FIELD_SIZE_ONE`, `GSF_FIELD_SIZE_TWO`, and `GSF_FIELD_SIZE_FOUR`. Once the field size has been set this value cannot be changed without rewriting the entire GSF file.

Table 2-1 GSF Beam Array Field Size Definitions

Array Subrecord	Data Representation	Size, bits	Scaled Dynamic Range
DEPTH	unsigned short (default)	16	0 to 65535
	unsigned int (option)	32	0 to 4294967295
NOMINAL_DEPTH	unsigned short (default)	16	0 to 65535
	unsigned int (option)	32	0 to 4294967295
ACROSS_TRACK	signed short (default)	16	-32768 to 32767
	signed int (option)	32	-2147483648 to 2147483647
ALONG_TRACK	signed short (default)	16	-32768 to 32767
	signed int (option)	32	-2147483648 to 2147483647
TRAVEL_TIME	unsigned short (default)	16	0 to 65535
	unsigned int (option)	32	0 to 4294967295
BEAM_ANGLE	signed short	16	-32768 to 32767
MEAN_CAL_AMPLITUDE	signed byte (default)	8	-128 to 127
	signed short (option)	16	-32768 to 32767
MEAN_REL_AMPLITUDE	unsigned byte (default)	8	0 to 255
	unsigned short (option)	16	0 to 65535
ECHO_WIDTH	unsigned byte (default)	8	0 to 255
	unsigned short (option)	16	0 to 65535
QUALITY_FACTOR	unsigned byte	8	0 to 255
RECEIVE_HEAVE	signed byte	8	-128 to 127
DEPTH_ERROR	unsigned short	16	0 to 65535
ACROSS_TRACK_ERROR	unsigned short	16	0 to 65535

ALONG_TRACK_ERROR	unsigned short	16	0 to 65535
QUALITY_FLAGS	unsigned byte	8	0 to 255
BEAM_FLAGS	unsigned byte	8	0 to 255
SIGNAL_TO_NOISE	signed byte	8	-128 to 127
BEAM_ANGLE_FORWARD	signed short	16	-32768 to 32767
VERTICAL_ERROR	unsigned short	16	0 to 65535
HORIZONTAL_ERROR	unsigned short	16	0 to 65535
SECTOR_NUMBER	unsigned byte	8	0 to 255
DETECTION_INFO	unsigned byte	8	0 to 255
INCIDENT_BEAM_ADJUSTMENT	signed byte	8	-128 to 127
SYSTEM_CLEANING	unsigned byte	8	0 to 255
DOPPLER_CORRECTION	signed byte	8	-128 to 127

Inputs:

sf a pointer to the *gsfScaleFactors* structure to be loaded
subrecordID the subrecord id for the beam array data
c_flag the compression flag for the beam array. This is a bit mask that combines the caller specified field size in the higher order four bits with the lower four bits reserved for future use to specify a compression algorithm. The supported field size values are defined as macros in *gsf.h* (*GSF_FIELD_SIZE_DEFAULT*, etc).
precision the precision to which the beam array data are to be stored(a value of 0.1 would indicate decimeter precision for depth)
offset the "DC" offset to scale the data by.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_CANNOT_REPRESENT_PRECISION
GSF_TOO_MANY_ARRAY_SUBRECORDS

2.2.6 Function: gsfGetScaleFactors

Usage:

```

int gsfGetScaleFactor(int          handle,
                     int          subrecordID,
                     unsigned char *c_flag,
                     double        *multiplier,
                     double        *offset)

```

Description:

gsfGetScaleFactors is used to obtain the beam array field size, compression flag, multiplier and DC offset values by which each swath bathymetry ping array subrecord is scaled.

gsfGetScalesFactors is called once for each array subrecord of interest. At least one swath

bathymetry ping record must have been read from, or written to, the file specified by handle prior to calling **gsfGetScaleFactors**.

Inputs:

Handle	the integer value set by a call to gsfOpen .
subrecordID	an integer value containing the subrecord id of the requested scale factors
c_flag	the address of an unsigned character to contain the optional beam array field size in the high order four bits, and the optional compression flag in the low order four bits. If the field size is not specified the default will be used. The high order four bits (beam_array_field_size) will be set to one of the following values: GSF_FIELD_SIZE_DEFAULT, GSF_FIELD_SIZE_ONE, GSF_FIELD_SIZE_TWO, or GSF_FIELD_SIZE_FOUR.
multiplier	the address of a double to contain the scaling multiplier
offset	the address of a double to contain the scaling DC offset.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_ILLEGAL_SCALE_FACTOR_MULTIPLIER
GSF_TOO_MANY_ARRAY_SUBRECORDS

2.2.7 Function: gsfSetDefaultScaleFactor

Usage:

```
int gsfSetDefaultScaleFactor(gsfSwathBathyPing *mb_ping)
```

Description:

gsfSetDefaultScaleFactor is a convenience function used to convert files stored in a vendor format to the gsf format. The function estimates reasonable scale factors for each of the arrays in the ping record. The function will estimate based on the default compression size and set the values of the ping's scale factors. This function requires some overhead as it will perform operations on each beam in each array contained in the ping record.

Inputs:

mb_ping	a pointer to the <i>gsfSwathBathyPing</i> which contains the beam arrays and will contain the estimated scale factors upon returning from the function.
---------	---

Returns:

The function returns 0 to indicate success.

Error Conditions:

None.

2.2.8 Function: gsfLoadDepthScaleFactorAutoOffset

Usage:

```
int gsfLoadDepthScaleFactorAutoOffset(gsfSwathBathyPing *ping,
                                      int subrecordID,
                                      int reset,
                                      double min_depth,
                                      double max_depth,
                                      double *last_corrector,
                                      char c_flag,
                                      double precision)
```

Description:

gsfLoadDepthScaleFactorAutoOffset may be used to load the scale factors for the depth subrecords of the swath bathymetry ping record scale factor structure. The function uses the tide and depth correction fields to help establish the offset component of the scale factor such that negative depth values may be supported. Negative depth values may be encountered when surveying above the tidal datum. In addition, this function may be used for systems mounted on subsea platforms where high depth precision may be supported even in deep water.

Inputs:

ping	a pointer to the <i>gsfSwathBathyPing</i> which contains the depth and tide correction values, and the scale factors data structure.
subrecordID	an integer value containing the subrecord ID for the beam array data; this must be either <code>GSF_SWATH_BATHY_SUBRECORD_DEPTH_ARRAY</code> , or <code>GSF_SWATH_BATHY_SUBRECORD_NOMINAL_DEPTH_ARRAY</code> .
reset	an integer value that will cause the internal logic to be refreshed when the value is non-zero; the first call to this function should use a non-zero reset, from then on, this value may be passed as zero.
min_depth	a double value that should be set to the minimum depth value contained in the depth array specified by subrecordID; this argument exists for completeness, but is currently not used.
max_depth	a double value that should be set to the maximum depth value contained in the depth array specified by subrecordID; when a depth threshold is exceeded, the offset used to support “signed depth” is no longer required and will no longer be used. This approach is necessary to avoid an integer overflow when the array data are scaled.
last_corrector	an address of a double value stored as permanent memory; successive calls to this function must pass the same address for this argument. This function will take care of setting the value at this address, but the caller is responsible for ensuring that the same permanent memory address is used for each call to this function.
C_flag	the compression flag for the beam array. This is a bit mask that combines

precision

the (optional) caller specified field size in the higher order four bits with the lower four bits reserved for future use to specify a compression algorithm. The supported field size values are defined as macros in `gsf.h` (`GSF_FIELD_SIZE_DEFAULT`, etc). See section 2.2.5 on **gsfLoadScaleFactors** for more information.
the precision to which the beam array data are to be stored (a value of 0.1 would indicate decimeter precision for depth).

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_UNRECOGNIZED_ARRAY_SUBRECORD_ID
GSF_CANNOT_REPRESENT_PRECISION
GSF_TOO_MANY_ARRAY_SUBRECORDS

2.2.9 Macro: `gsfTestPingStatus`

Usage:

```
unsigned short gsfTestPingStatus(ping_flags, usflag)
```

Description:

This function returns the value of a single flag within the `ping_flags` field of the *gsfSwathBathymetry* record

Inputs:

`ping_flags` The contents of the `ping_flags` field.
`usflag` An unsigned short integer with a single bit set to identify the flag being tested.

Returns:

This macro TRUE if the bit within `ping_flags`, which corresponds to the bit set in `usflags`, is set. Otherwise, the macro returns FALSE.

Error Conditions:

None

2.2.10 Macro: `gsfSetPingStatus`

Usage:

```
unsigned short gsfSetPingStatus(ping_flags, usflag)
```

Description:

This function sets a bit within the within the `ping_flags` field of the *gsfSwathBathymetry* record

Inputs:

`ping_flags` The original contents of the `ping_flags` field.
`usflag` An unsigned short integer with a single bit set to identify the flag to be set.

Returns:

A new copy of the `ping_flags` field with the corresponding bit set.

Error Conditions:

None

2.2.11 Macro: `gsfClearPingStatus`

Usage:

```
unsigned short gsfClearPingStatus(ping_flags, usflag)
```

Description:

This function clears a bit within the within the `ping_flags` field of the *gsfSwathBathymetry* record.

Inputs:

`ping_flags` The original contents of the `ping_flags` field.
`usflag` An unsigned short integer with a single bit set to identify the flag to be cleared.

Returns:

A new copy of the `ping_flags` field with the corresponding bit cleared.

Error Conditions:

None

2.3 Information Functions

Information functions include those that

- decode error conditions,
- return the time associated with a record at a specific location,
- return the location of the file pointer as a percentage of the total file size,
- provide the number and types of records within a file,
- provide information about beam widths of various types of sonar data and,
- for sonars with two transducers, determine whether a specific data record is from the starboard or port transducer.
- provide the name of the sensor

2.3.1 Function: `gsfPrintError`

Usage:

```
void gsfPrintError(FILE * fp)
```

Description:

This function prints a short message describing the most recent error encountered. Call this function if a -1 is returned from one of the GSF functions.

Inputs:

fp a pointer to a FILE to which the message is written.

Returns:

None

Error Conditions:

None

2.3.2 Function: gsfStringError

Usage:

```
char *gsfStringError(void);
```

Description:

This function returns a short message describing the most recent error encountered. Call this function if a -1 is returned from one of the gsf functions.

Inputs:

None

Returns:

Pointer to a string containing the text message.

Error Conditions:

None

2.3.3 Function: gsfIndexTime

Usage:

```
int gsfIndexTime(int    handle,  
                 int     record_type,  
                 int     record_number,  
                 time_t  *sec,  
                 long    *nsec)
```

Description:

This function returns the time associated with a specified record number and type. It also returns the record number that was read.

Inputs:

handle	GSF file handle assigned by gsfOpen or gsfOpenBuffered
record_type	record type to be retrieved
record_number	record number to be retrieved (Setting this argument to -1 will get the time and record number of the last record of type <code>record_type</code>)
sec	Seconds since the beginning of the epoch (as defined in the GSF processing parameter record.)
nsec	Nanoseconds since the beginning of the second.

Returns:

This function returns the record number if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_FILE_SEEK_ERROR
GSF_INDEX_FILE_READ_ERROR
GSF_RECORD_TYPE_NOT_AVAILABLE

2.3.4 Function: **gsfPercent**

Usage:

```
int gsfPercent (int handle)
```

Description:

This function returns the location of the file pointer expressed as a percentage of the total file size. It may obtain an indication of how far along a program is in reading a GSF data file. The file size is obtained when the file is opened. If the file is being updated by another program, the value returned will be in error and will reflect the percentage based on the file's size at the time that calling program opened the file.

Inputs:

handle	gsf file handle assigned by gsfOpen or gsfOpenBuffered
--------	--

Returns:

This function returns the current file position as a percentage of the file size, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_FILE_TELL_ERROR

2.3.5 Function: **gsfGetNumberRecords**

Usage:

```
int gsfGetNumberRecords (int handle,  
                          int desiredRecord)
```

Description:

This function returns the number of records of a given type. The number of records is retrieved from the index file, so the file must have been opened for direct access (*GSF_READONLY_INDEX*, or *GSF_UPDATE_INDEX*).

Inputs:

handle	the handle to the file as provided by gsfOpen
desiredRecord	the desired record or <i>GSF_NEXT_RECORD</i>

Returns:

This function returns the number of records of type *desiredRecord* contained in the GSF file designated by handle, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_BAD_ACCESS_MODE

2.3.6 Function: gsfGetSwathBathyBeamWidths

Usage:

```
int gsfGetSwathBathyBeamWidths(gsfRecords *data,  
                                double      *fore_aft,  
                                double      *athwartship)
```

Description:

This function returns to the caller the fore-aft and the port-starboard beam widths in degrees for a swath bathymetry multibeam sonar, given a *gsfRecords* data structure containing a populated *gsfSwathBathyPing* structure.

Inputs:

data	The address of a <i>gsfRecords</i> data structure maintained by the caller which contains a populated <i>gsfSwathBathyPing</i> substructure.
fore_aft	The address of a double allocated by the caller which will be loaded with the sonar's fore/aft beam width in degrees. A value of <i>GSF_BEAM_WIDTH_UNKNOWN</i> is used when the beam width is not known.
athwartship	The address of a double allocated by the caller which will be loaded with the sonar's athwartship beam width in degrees. A value of <i>GSF_BEAM_WIDTH_UNKNOWN</i> is used when the beam width is not known.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

None.

2.3.7 Function: gsfGetSwathBathyArrayMinMax

Usage:

```
int gsfGetSwathBathyArrayMinMax(gsfSwathBathyPing *ping,
                                int subrecordID,
                                double *min_value,
                                double *max_value)
```

Description:

This function returns to the caller the minimum and maximum supportable values for each of the swath bathymetry arrays. The minimum and maximum values are determined based on the scale factors and the array type.

Inputs:

ping	The address of a <i>gsfSwathBathyPing</i> data structure that contains the depth and tide correction values, as well as the scale factors data structure.
subrecordID	The subrecord ID for the beam array data.
min_value	The address of a double value allocated by the caller into which will be placed the minimum value that may be represented for this array type.
max_value	The address of a double value allocated by the caller into which will be placed the maximum value that may be represented for this array type.

Returns:

This function returns zero if successful, or -1 if an error occurred. *gsfError* is set to indicate the error.

Error Conditions:

GSF_UNRECOGNIZED_ARRAY_SUBRECORD_ID
GSF_ILLEGAL_SCALE_FACTOR_MULTIPLIER

2.3.8 Function: gsfIsStarboardPing

Usage:

```
int gsfIsStarboardPing(gsfRecords *data)
```

Description:

This function uses the sonar specific portion of a *gsfSwathBathymetry* ping structure to determine if the ping is from the starboard arrays of a multibeam installation with dual transducers.

Inputs:

data The address of a *gsfRecords* data structure maintained by the caller containing a populated *gsfSwathBathyPing* substructure.

Returns:

This function returns non-zero if the ping contained in the passed data represents a starboard looking ping from a dual headed sonar installation. Otherwise, zero is returned. If the sonar does not have dual transducers, a value of zero will be returned.

Error Conditions:

None

2.3.9 Function: *gsf_register_progress_callback*

Usage:

```
void gsf_register_progress_callback(GSF_PROGRESS_CALLBACK progressCB)
```

Description:

This function registers a callback function, defined by the user, to be called to report the progress of the index file creation. If no progress callback is registered, status is printed to stdout if the *DISPLAY_SPINNER* macro is defined during compilation of the GSF library.

Inputs:

progressCB The name of the progress callback function to call when creating the GSF index file. The progress callback will accept two integer arguments, and this function will be called whenever the percent complete changes. This first argument will be one of the following three values, to represent the state of the progress:

- 1 = Reading GSF file
- 2 = Creating new index file
- 3 = Appending to existing index file

The second argument contains the percent complete of the current state.

Returns:

None

Error Conditions:

None

2.3.10 Function: *gsfGetSonarTextName*

Usage:

```
char *gsfGetSonarTextName(gsfSwathBathyPing *ping)
```

Description:

This function returns the name of the sensor based on the sensor id contained in the ping structure.

Inputs:

Ping The address of a *gsfSwathBathyPing* data structure that contains the *sensor_id* value, as well as the mode value (mode is used for the Reson SeaBat 9001, 9002, and 9003)

Returns:

Pointer to a string containing the sensor name, or “Unknown” if the sensor id is not defined.

Error Conditions:

None

2.3.11 Function: gsfFileSupportsRecalculateXYZ

Usage: `int gsfFileSupportsRecalculateXYZ(int handle, int *status)`

Description: This function reads the GSF file referenced by handle and determines if the file contains sufficient information to support a full recalculation of the platform relative XYZ values from raw measurements. This function rewinds the file to the first record and reads through the file looking for the information required to support a full swath recalculation from raw measurements and supporting navigation, attitude, SVP and installation offset information. On success, the file pointer is reset to the beginning of the file before the function returns.

Inputs:

handle GSF file handle assigned by **gsfOpen** or **gsfOpenBuffered**
status A pointer to an integer allocated by caller into which the function result is placed.
 *status is assigned a value of 1 if this file provides sufficient information to support full recalculation of the platform relative XYZ values, otherwise *status is assigned a value of 0.

Returns: This function returns zero if successful or -1 if an error occurred.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR
GSF_READ_TO_END_OF_FILE
GSF_READ_ERROR
GSF_RECORD_SIZE_ERROR
GSF_INSUFFICIENT_SIZE
GSF_CHECKSUM_FAILURE
GSF_UNRECOGNIZED_RECORD_ID
GSF_HEADER_RECORD_DECODE_FAILED
GSF_SVP_RECORD_DECODE_FAILED
GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
GSF_SENSOR_PARAM_RECORD_DECODE_FAILED

GSF_COMMENT_RECORD_DECODE_FAILED
GSF_HISTORY_RECORD_DECODE_FAILED
GSF_NAV_ERROR_RECORD_DECODE_FAILED

2.3.12 Function: **gsfFileSupportsRecalculateTPU**

Usage: `int gsfFileSupportsRecalculateTPU(int handle, int *status)`

Description: This function reads the GSF file referenced by handle and determines if the file contains sufficient information to support calculation of the total propagated uncertainty (TPU) values. This function rewinds the file to the first record and reads through the file looking for the information required to support calculation of vertical and horizontal propagated uncertainty. The total propagated uncertainty arrays are the horizontal_error and the vertical_error beam arrays. On success, the file pointer is reset to the beginning of the file before the function returns.

Inputs:

Handle	GSF file handle assigned by gsfOpen or gsfOpenBuffered
Status	A pointer to an integer allocated by caller into which the function result is placed. *status is assigned a value of 1 if this file provides sufficient information to support calculation of the total propagated uncertainty array values, otherwise *status is assigned a value of 0.

Returns: This function returns zero if successful or -1 if an error occurred.

Error Conditions:

GSF_BAD_FILE_HANDLE
GSF_FILE_SEEK_ERROR
GSF_FLUSH_ERROR
GSF_READ_TO_END_OF_FILE
GSF_READ_ERROR
GSF_RECORD_SIZE_ERROR
GSF_INSUFFICIENT_SIZE
GSF_CHECKSUM_FAILURE
GSF_UNRECOGNIZED_RECORD_ID
GSF_HEADER_RECORD_DECODE_FAILED
GSF_SVP_RECORD_DECODE_FAILED
GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
GSF_COMMENT_RECORD_DECODE_FAILED
GSF_HISTORY_RECORD_DECODE_FAILED
GSF_NAV_ERROR_RECORD_DECODE_FAILED

2.3.13 Function: **gsfFileSupportsRecalculateNominalDepth**

Usage: `int gsfFileSupportsRecalculateNominalDepth(int handle, int *status)`

Description: This function reads the GSF file referenced by handle and determines if the file contains sufficient information to support calculation of the nominal depth array. This function rewinds the file to the first record and reads through the file looking for the information required to support calculation of the optional nominal depth array. The nominal depth values represent the depth relative to a sound speed of 1500 meters second. On success, the file pointer is reset to the beginning of the file before the function returns.

Inputs:

handle	GSF file handle assigned by gsfOpen or gsfOpenBuffered
status	A pointer to an integer allocated by caller into which the function result is placed. *status is assigned a value of 1 if this file provides sufficient information to support calculation of the nominal depth array, otherwise *status is assigned a value of 0.

Returns: This function returns zero if successful or -1 if an error occurred.

Error Conditions:

- GSF_BAD_FILE_HANDLE
- GSF_FILE_SEEK_ERROR
- GSF_FLUSH_ERROR
- GSF_READ_TO_END_OF_FILE
- GSF_READ_ERROR
- GSF_RECORD_SIZE_ERROR
- GSF_INSUFFICIENT_SIZE
- GSF_CHECKSUM_FAILURE
- GSF_UNRECOGNIZED_RECORD_ID
- GSF_HEADER_RECORD_DECODE_FAILED
- GSF_SVP_RECORD_DECODE_FAILED
- GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
- GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
- GSF_COMMENT_RECORD_DECODE_FAILED
- GSF_HISTORY_RECORD_DECODE_FAILED
- GSF_NAV_ERROR_RECORD_DECODE_FAILED

2.3.14 Function: **gsfFileContainsMBAmpitude**

Usage: `int gsfFileContainsMBAmpitude(int handle, int *status)`

Description: This function reads the GSF file referenced by handle and determines if the file contains the average per receive beam amplitude data. This function rewinds the file to the first record and reads through the file up to and including the first ping record. If amplitude data are contained in the first ping record it is assumed that amplitude data are contained with all ping records in this file. On success, the file pointer is reset to the beginning of the file before the function returns.

Inputs:

handle	GSF file handle assigned by gsfOpen or gsfOpenBuffered
status	A pointer to an integer allocated by caller into which the function result is placed. *status is assigned a value of 1 if this file contains the optional per-receive-beam average amplitude beam array, otherwise *status is assigned a value of 0.

Returns: This function returns zero if successful or -1 if an error occurred.

Error Conditions:

GSF_BAD_FILE_HANDLE
 GSF_FILE_SEEK_ERROR
 GSF_FLUSH_ERROR
 GSF_READ_TO_END_OF_FILE
 GSF_READ_ERROR
 GSF_RECORD_SIZE_ERROR
 GSF_INSUFFICIENT_SIZE
 GSF_CHECKSUM_FAILURE
 GSF_UNRECOGNIZED_RECORD_ID
 GSF_HEADER_RECORD_DECODE_FAILED
 GSF_SVP_RECORD_DECODE_FAILED
 GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
 GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
 GSF_COMMENT_RECORD_DECODE_FAILED
 GSF_HISTORY_RECORD_DECODE_FAILED
 GSF_NAV_ERROR_RECORD_DECODE_FAILED

2.3.15 Function: gsfFileContainsMBImagery

Usage: int gsfFileContainsMBImagery(int handle, int *status)

Description: This function reads the GSF file referenced by handle and determines if the file contains the per-receive-beam imagery time series data. This function rewinds the file to the first record and reads through the file up to and including the first ping record. If MB imagery data are contained in the first ping record it is assumed that MB imagery data are contained with all ping records in this file. On success, the file pointer is reset to the beginning of the file before the function returns.

Inputs:

handle	GSF file handle assigned by gsfOpen or gsfOpenBuffered
status	A pointer to an integer allocated by caller into which the function result is placed. *status is assigned a value of 1 if this file contains the optional per-receive-beam imagery time series data, otherwise *status is assigned a value of 0.

Returns: This function returns zero if successful or -1 if an error occurred.

Error Conditions:

GSF_BAD_FILE_HANDLE

GSF_FILE_SEEK_ERROR
 GSF_FLUSH_ERROR
 GSF_READ_TO_END_OF_FILE
 GSF_READ_ERROR
 GSF_RECORD_SIZE_ERROR
 GSF_INSUFFICIENT_SIZE
 GSF_CHECKSUM_FAILURE
 GSF_UNRECOGNIZED_RECORD_ID
 GSF_HEADER_RECORD_DECODE_FAILED
 GSF_SVP_RECORD_DECODE_FAILED
 GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
 GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
 GSF_COMMENT_RECORD_DECODE_FAILED
 GSF_HISTORY_RECORD_DECODE_FAILED
 GSF_NAV_ERROR_RECORD_DECODE_FAILED

2.3.16 Function: **gsfIsNewSurveyLine**

Usage: `int gsfIsNewSurveyLine (int handle, gsfRecords *rec, double azimuth_change, double *last_heading)`

Description: This function provides an approach for calling applications to determine if the last ping read from a GSF file is from the same survey transect line, or if the last ping is from a newly started survey line. The implementation looks for a change in platform heading to determine that the last ping read is from a new survey line. External to this function, calling applications can decide on their own if the first ping read from a newly opened GSF file should be considered to be from a new survey transect line or not. This function assumes that the GSF file is read in chronological order from the beginning of the file, file access can be either direct or sequential

Inputs:

<code>handle</code>	GSF file handle assigned by gsfOpen or gsfOpenBuffered
<code>rec</code>	The address of a <i>gsfRecords</i> data structure maintained by the caller which contains a populated <i>gsfSwathBathyPing</i> substructure obtained from recent call to <code>gsfRead</code> .
<code>azimuth_change</code>	A trigger value set by the calling application to be used as the threshold for detecting the end heading change associated with the end of a survey line.
<code>last_heading</code>	The address of a double allocated by the calling that is set by <code>gsfIsNewSurveyLine</code> when a new line is detected. The application program should allocate this double such that it's memory persists for all calls to <code>gsfIsNewSurveyLine</code> . The function depends on this value persisting from one call to the next.

Returns: This function returns zero when ping is not considered to be from a new survey line and non-zero when the ping is considered to be from a new survey line.

Error Conditions:

GSF_BAD_FILE_HANDLE
 GSF_FILE_SEEK_ERROR

GSF_FLUSH_ERROR
GSF_READ_TO_END_OF_FILE
GSF_READ_ERROR
GSF_RECORD_SIZE_ERROR
GSF_INSUFFICIENT_SIZE
GSF_CHECKSUM_FAILURE
GSF_UNRECOGNIZED_RECORD_ID
GSF_HEADER_RECORD_DECODE_FAILED
GSF_SVP_RECORD_DECODE_FAILED
GSF_PROCESS_PARAM_RECORD_DECODE_FAILED
GSF_SENSOR_PARAM_RECORD_DECODE_FAILED
GSF_COMMENT_RECORD_DECODE_FAILED
GSF_HISTORY_RECORD_DECODE_FAILED
GSF_NAV_ERROR_RECORD_DECODE_FAILED

3. ERROR CODE DESCRIPTIONS

Any GSF function that returns an error code also sets the value of *gsfError* before returning. Table 3-1 lists the reasons for error. **gsfPrintError** or **gsfStringError** can be used to generate a text string of the reason for the error.

Note that the current version of GSFLib does provide text string translations for all error code returns; however, not all definitions have unique values. A future release will address this issue. Table 3-1 presents all the reasons supported by **gsfPrintError**. The following table is a complete listing of all error return codes.

Table 3-1 GSF Error Codes

Value of <i>gsfError</i>	Value	Reason for error
<i>GSF_ATTITUDE_RECORD_DECODE_FAILED</i>	-49	"GSF Error decoding attitude record"
<i>GSF_ATTITUDE_RECORD_ENCODE_FAILED</i>	-50	
<i>GSF_BAD_ACCESS_MODE</i>	-3	"GSF Error illegal access mode"
<i>GSF_BAD_FILE_HANDLE</i>	-24	"GSF Error bad file handle"
<i>GSF_BAD_SEEK_OPTION</i>	-15	"GSF Error unrecognized file seek option"
<i>GSF_CANNOT_REPRESENT_PRECISION</i>	-22	"GSF Error illegal scale factor multiplier specified"
<i>GSF_CHECKSUM_FAILURE</i>	-8	"GSF Error data checksum failure"
<i>GSF_COMMENT_RECORD_DECODE_FAILED</i>	-30	"GSF Error decoding comment record"
<i>GSF_COMMENT_RECORD_ENCODE_FAILED</i>	-30	
<i>GSF_CORRUPT_INDEX_FILE_ERROR</i>	-37	"GSF Error index file is corrupted, delete index file"
<i>GSF_FILE_CLOSE_ERROR</i>	-9	"GSF Error closing gsf file"
<i>GSF_FILE_SEEK_ERROR</i>	-16	"GSF Error file seek failed"
<i>GSF_FILE_TELL_ERROR</i>	-35	"GSF Error file tell failed"
<i>GSF_FLUSH_ERROR</i>	-34	"GSF Error flushing data buffers(s)"
<i>GSF_FOPEN_ERROR</i>	-1	"GSF Unable to open requested file"
<i>GSF_HEADER_RECORD_DECODE_FAILED</i>	-25	"GSF Error decoding header record"
<i>GSF_HEADER_RECORD_ENCODE_FAILED</i>	-25	
<i>GSF_HISTORY_RECORD_DECODE_FAILED</i>	-31	"GSF Error decoding history record"
<i>GSF_HISTORY_RECORD_ENCODE_FAILED</i>	-31	
<i>GSF_HV_NAV_ERROR_RECORD_DECODE_FAILED</i>	-48	"GSF Error decoding horizontal/vertical navigation error record"
<i>GSF_HV_NAV_ERROR_RECORD_ENCODE_FAILED</i>	-47	"GSF Error encoding horizontal/vertical navigation error record"
<i>GSF_ILLEGAL_SCALE_FACTOR_MULTIPLIER</i>	-21	"GSF Error illegal scale factor multiplier specified"
<i>GSF_INDEX_FILE_OPEN_ERROR</i>	-36	"GSF Error open of index file failed"
<i>GSF_INDEX_FILE_READ_ERROR</i>	-44	"GSF Error index file read error"
<i>GSF_INSUFFICIENT_SIZE</i>	-6	"GSF Error insufficient size specified"
<i>GSF_INVALID_NUM_BEAMS</i>	-42	"GSF Error invalid number of beams"
<i>GSF_INVALID_RECORD_NUMBER</i>	-43	"GSF Error invalid record number"
<i>GSF_MB_PING_RECORD_DECODE_FAILED</i>	-26	"GSF Error decoding multibeam ping record"
<i>GSF_MB_PING_RECORD_ENCODE_FAILED</i>	-26	
<i>GSF_MEMORY_ALLOCATION_FAILED</i>	-12	"GSF Error memory allocation failure"
<i>GSF_NAV_ERROR_RECORD_DECODE_FAILED</i>	-32	"GSF Error decoding latitude/longitude navigation error record"
<i>GSF_NAV_ERROR_RECORD_ENCODE_FAILED</i>	-32	

<i>GSF_NORMAL</i>	0	
<i>GSF_OPEN_TEMP_FILE_FAILED</i>	-51	"GSF Failed to open temporary file for index creation"
<i>GSF_PARAM_SIZE_FIXED</i>	-45	"GSF Error unable to update existing file with increased record size"
<i>GSF_PROCESS_PARAM_RECORD_DECODE_FAILED</i>	-28	"GSF Error decoding processing parameters record"
<i>GSF_PROCESS_PARAM_RECORD_ENCODE_FAILED</i>	-28	
<i>GSF_READ_ERROR</i>	-4	"GSF Error reading input data"
<i>GSF_READ_TO_END_OF_FILE</i>	-23	
<i>GSF_RECORD_SIZE_ERROR</i>	-7	"GSF Error record size is out of bounds"
<i>GSF_RECORD_TYPE_NOT_AVAILABLE</i>	-39	"GSF Error requested indexed record type not in gsf file"
<i>GSF_SCALE_INDEX_CALLOC_ERROR</i>	-38	"GSF Error calloc of scale factor index memory failed"
<i>GSF_SENSOR_PARAM_RECORD_DECODE_FAILED</i>	-29	"GSF Error decoding sensor parameters record"
<i>GSF_SENSOR_PARAM_RECORD_ENCODE_FAILED</i>	-29	
<i>GSF_SETVBUF_ERROR</i>	-33	"GSF Error setting internal file buffering"
<i>GSF_SINGLE_BEAM_ENCODE_FAILED</i>	-46	"GSF Error stream encode failure"
<i>GSF_STREAM_DECODE_FAILURE</i>	-14	"GSF Error stream decode failure"
***Note: error code is not used		
<i>GSF_SUMMARY_RECORD_DECODE_FAILED</i>	-40	"GSF Error decoding summary record"
<i>GSF_SUMMARY_RECORD_ENCODE_FAILED</i>	-41	"GSF Error encoding summary record"
<i>GSF_SVP_RECORD_DECODE_FAILED</i>	-27	"GSF Error decoding SVP record"
<i>GSF_SVP_RECORD_ENCODE_FAILED</i>	-27	
<i>GSF_TOO_MANY_ARRAY_SUBRECORDS</i>	-10	"GSF Error too many array subrecords"
<i>GSF_TOO_MANY_OPEN_FILES</i>	-11	"GSF Error too many open files"
<i>GSF_UNRECOGNIZED_ARRAY_SUBRECORD_ID</i>	-19	"GSF Error unrecognized array subrecord id "
<i>GSF_UNRECOGNIZED_DATA_RECORD</i>	-18	"GSF Error unrecognized data record id"
<i>GSF_UNRECOGNIZED_FILE</i>	-2	"GSF Error unrecognized file"
<i>GSF_UNRECOGNIZED_RECORD_ID</i>	-13	"GSF Error unrecognized record id"
<i>GSF_UNRECOGNIZED_SENSOR_ID</i>	-17	"GSF Error unrecognized sensor specific subrecord id"
<i>GSF_UNRECOGNIZED_SUBRECORD_ID</i>	-20	"GSF Error unrecognized subrecord id"
<i>GSF_WRITE_ERROR</i>	-5	"GSF Error writing output data"
<i>Unrecognized error condition</i>		"GSF unknown error"

4. C-LANGUAGE DEFINITIONS OF STRUCTURES USED BY GSFLIB

GSFLib is built upon several complex data structures that are passed to applications using the library to access data. This section describes these complex data structures.

4.1 Definition of GSF Data Records

Eleven data records define GSF data. Subsequent sections define each of these records. The `gsfRecords` structure allows all records to be addressed as a unit.

```
typedef struct t_gsfRecords
{
    gsfHeader          header;
    gsfSwathBathySummary summary;
    gsfSwathBathyPing  mb_ping;
    gsfSingleBeamPing  sb_ping;
    gsfSVP              svp;
    gsfProcessingParameters process_parameters;
    gsfSensorParameters sensor_parameters;
    gsfComment          comment;
    gsfHistory          history;
    gsfNavigationError  nav_error;
    gsfHVNavigationError hv_nav_error;
    gsfAttitude         attitude;
} gsfRecords;
```

4.1.1 Header Record

A header record is required to be the first record of every GSF data file.

```
#define GSF_VERSION_SIZE 12
typedef struct t_gsfHeader
{
    char          version[GSF_VERSION_SIZE];
}
gsfHeader;
```

4.1.2 Swath Bathymetry Ping Record

```
typedef struct t_gsfSwathBathyPing
{
    struct timespec  ping_time;          /* seconds and nanoseconds */
    double           latitude;           /* in degrees, north is positive */
    double           longitude;          /* in degrees, west is positive */
    double           height;             /* height above ellipsoid */
    double           sep;                /* ellipsoid to chart datum */
    short            number_beams;        /* in this ping */
    short            center_beam;        /* offset into array (0 = portmost outer) */
    unsigned short   ping_flags;         /* flags to mark status of this ping */
    short            reserved;           /* for future use */
    double           tide_corrector;      /* in meters */
    double           gps_tide_corrector; /* in meters */
    double           depth_corrector;     /* in meters */
    double           heading;            /* in degrees */
    double           pitch;              /* in degrees */
    double           roll;               /* in degrees */
    double           heave;             /* in meters */
}
```

```

double      course;          /* in degrees */
double      speed;           /* in knots */
gsfScaleFactors scaleFactors; /* The array scale factors for this data */
double      *depth;          /* depth array (meters) */
double      *nominal_depth;  /* Array of depth relative to 1500 m/s */
double      *across_track;   /* across track array (meters) */
double      *along_track;    /* along track array (meters) */
double      *travel_time;    /* roundtrip travel time array (seconds) */
double      *beam_angle;     /* beam angle array degrees from vertical */
double      *mc_amplitude;    /* mean, calibrated beam amplitude array (dB
                             re 1V/micro pascal at 1 meter) */
double      *mr_amplitude;    /* mean, relative beam amplitude array (dB
                             re 1V/micro pascal at 1 meter) */
double      *echo_width;     /* echo width array (seconds) */
double      *quality_factor; /* quality factor array (dimensionless) */
double      *receive_heave;  /* Array of heave data (meters) */
double      *depth_error;    /* Array of estimated vertical error
                             (meters) */
double      *across_track_error; /* Array of estimated across track error
                             (meters) */
double      *along_track_error; /* Array of estimated along track error
                             (meters) */
unsigned char *quality_flags; /* Two bit beam detection flags provided by
                             Reson sonar */
unsigned char *beam_flags;    /* Array of beam status flags */
double      *signal_to_noise; /* signal to noise ratio (dB) */
double      *beam_angle_forward; /* beam angle forward array (degrees
                             counterclockwise from stbd.) */
double      *vertical_error;  /* Array of estimated vertical error
                             (meters, at 95% confidence) */
double      *horizontal_error; /* Array of estimated horizontal error
                             (meters, at 95% confidence) */
unsigned short *sector_number; /* Array of values that specify the transit
                             sector for this beam */
unsigned short *detection_info; /* Array of values that specify the method
                             of bottom detection */
double      *incident_beam_adj; /* Array of values that specify incident
                             beam angle adjustment from beam_angle */
unsigned short *system_cleaning; /* Array of values that specify data
                             cleaning information from the sensor
                             system */
double      *doppler_corr;    /* Array of values used to correct the
                             travel times for Doppler when
                             transmission is FM */
int          sensor_id;       /* a definition which specifies the sensor */
gsfSensorSpecific sensor_data; /* union of known sensor specific data */
gsfBRBIntensity *brb_inten;   /* Structure containing bathymetric receive
                             beam time series intensities */
}
gsfSwathBathyPing;

```

4.1.2.1 Scale Factor Subrecord

```

typedef struct t_gsfScaleInfo
{
    unsigned char  compressionFlag; /* Specifies bytes of storage in high order nibble
                                     and type of compression in low order nibble */
    double         multiplier;      /* the scale factor (millionths) for the array */
    double         offset;          /* dc offset to scale data by */
} gsfScaleInfo;

typedef struct t_gsfScaleFactors
{

```

```

        int          numArraySubrecords; /* number of scaling factors we actually have */
        gsfScaleInfo scaleTable[GSF_MAX_PING_ARRAY_SUBRECORDS];
    } gsfScaleFactors;

```

4.1.2.2 Multibeam Sensor-specific Subrecords

```

/* Define the typeIII specific data structure */
typedef struct t_gsfTypeIIISpecific
{
    unsigned short  leftmost_beam; /* 0 - leftmost possible beam */
    unsigned short  rightmost_beam;
    unsigned short  total_beams;
    unsigned short  nav_mode;
    unsigned short  ping_number;
    unsigned short  mission_number;
}
t_gsfTypeIIISpecific;

/* The gsfCmpSassSpecific data structure is intended to replace the gsfTypeIII Specific
 * data structure in a future release. All new coding should use the gsfCmpSassSpecific
 * data structure.
 */

/* Define the CMP (Compressed) SASS specific data structure (from sass.h) */
typedef struct t_gsfCmpSassSpecific
{
    /*****
    *
    * Mapping from Compressed SASS (BOSDAT) to GSF record
    *
    * from          to          comment
    * =====
    *
    * lntens         ping.heave   mapped only when year is post 1991 or
    *                                     user has elected to force mapping.
    * lfreq          not-mapped
    * ldraft         comment     APPLIED_DRAFT comment record
    * svp.svel       svp.sound_velocity at <= 1000 ... FATHOMS
    *                                     at <= 2500 ... METERS
    *                                     otherwise ... FEET
    * svp.depthl     svp.depth    (see sound_velocity)
    * lmishn         comment     MISSION_NUMBER comment record
    * luyr           ping_time    GSF time record from 1960 to 1970 base
    * pitchl         ping.pitch
    * rolll         ping.roll
    * lbear          ping.heading SASS specific (not Seabeam)
    * pinhd          ping.heading Seabeam specific (not SASS)
    * depth          ping.nominal_depth FATHOMS_TO_METERS_NOMINAL
    * pslatl         ping.across_track YARDS_TO_METERS_EXACT
    * bltime         ping.travel_time
    * ampl           ping.mr_amplitude
    * <ftaf file>    ping.beam_flags  HMPS_FLAGS
    * alpos          ping.along_track SASS specific YARDS_TO_METERS_EXACT
    *
    *****/

    double lfreq; /* sea-surface sound velocity in feet/sec from bosdat(lfreq) */
    double lntens; /* since 1992 this value has represented the heave associated with
                    the ping; prior to 1992, field description unknown */
}
t_gsfCmpSassSpecific;

```

```

/* Define the 16 Beam SeaBeam specific data structure */
typedef struct t_gsfSeabeamSpecific
{
    unsigned short  EclipseTime; /* In 10ths of seconds */
}
t_gsfSeabeamSpecific;

typedef struct t_gsfSBampSpecific
{
    unsigned char    hour;
    unsigned char    minute;
    unsigned char    second;
    unsigned char    hundredths;
    unsigned int     block_number;
    short            avg_gate_depth;
}
t_gsfSBampSpecific;

/* Define the Seemap specific data structure */
typedef struct t_gsfSeemapSpecific
{
    double           portTransmitter[2];
    double           stbdTransmitter[2];
    double           portGain;
    double           stbdGain;
    double           portPulseLength;
    double           stbdPulseLength;
    double           pressureDepth;
    double           altitude;
    double           temperature;
}
t_gsfSeemapSpecific;

/* Define the EM950/EM1000 specific data structure */
typedef struct t_gsfEM950Specific
{
    int              ping_number;
    int              mode;
    int              ping_quality;
    double           ship_pitch;
    double           transducer_pitch;
    double           surface_velocity;
}
t_gsfEM950Specific;

/* Define the EM100 specific data structure */
typedef struct t_gsfEM100Specific
{
    double           ship_pitch;
    double           transducer_pitch;
    int              mode;
    int              power;
    int              attenuation;
    int              tvlg;
    int              pulse_length;
    int              counter;
}
t_gsfEM100Specific;

/* Define the EM121A specific data structure */
typedef struct t_gsfEM121ASpecific
{
    int              ping_number;

```

```

    int         mode;
    int         valid_beams;
    int         pulse_length;
    int         beam_width;
    int         tx_power;
    int         tx_status;
    int         rx_status;
    double      surface_velocity;
}
t_gsfEM121ASpecific;

/* Define a data structure to hold the Simrad EM3000 series run time parameters. */
typedef struct t_gsfEM3RunTime
{
    int         model_number;           /* from the run-time parameter datagram */
    struct timespec dg_time;           /* from the run-time parameter datagram */
    int         ping_number;           /* sequential counter 0 - 65535 */
    int         serial_number;         /* The sonar head serial number */
    int         system_status;         /* normally = 0 */
    int         mode;                 /* 0=nearfield, 1=normal, 2=target,
                                     3=deep, 4=very deep */

    int         filter_id;
    double      min_depth;             /* meters */
    double      max_depth;             /* meters */
    double      absorption;            /* dB/km */
    double      pulse_length;          /* micro seconds */
    double      transmit_beam_width;   /* degrees */
    int         power_reduction;       /* dB */
    double      receive_beam_width;    /* degrees */
    int         receive_bandwidth;     /* Hz */
    int         receive_gain;          /* dB */
    int         cross_over_angle;      /* degrees */
    int         ssv_source;            /* 0=sensor, 1>manual, 2=profile */
    int         swath_width;           /* total swath width in meters */
    int         beam_spacing;          /* 0=beamwidth, 1=equiangle,
                                     2=equidistant, 3=intermediate */

    int         coverage_sector;       /* total coverage in degrees */
    int         stabilization;
    int         port_swath_width;       /* maximum port swath width in meters */
    int         stbd_swath_width;      /* maximum starboard swath width in
                                     meters */

    int         port_coverage_sector;  /* maximum port coverage in degrees */
    int         stbd_coverage_sector;  /* maximum starboard coverage in degrees */
    int         hilo_freq_absorp_ratio;
    int         spare1;                /* four spare bytes */
}
t_gsfEM3RunTime;

/* Define the Simrad EM3000 series specific data structure */
typedef struct t_gsfEM3Specific
{
    /* The first nine values are updated with each depth datagram */
    int         model_number;          /* ie: 3000, ... */
    int         ping_number;           /* 0 - 65535 */
    int         serial_number;         /* 100 - 65535 */
    double      surface_velocity;      /* in m/s */
    double      transducer_depth;      /* transmit transducer depth in meters */
    int         valid_beams;           /* number of valid beams for this ping */
    int         sample_rate;           /* in Hz */
    double      depth_difference;       /* in meters between sonar heads in em3000d
                                     configuration */

    int         offset_multiplier;     /* transducer depth offset multiplier */
}
/* The gsfEM3RunTime data structure is updated with each run-time parameter datagram*/

```

```

    gsfEM3RunTime run_time[2]; /* A two element array is needed to support em3000d */
}
t_gsfEM3Specific;

/* Define the Reson SeaBat specific data structure */
typedef struct t_gsfSeaBatSpecific
{
    int         ping_number;
    double      surface_velocity;
    int         mode;
    int         sonar_range;
    int         transmit_power;
    int         receive_gain;
}
t_gsfSeaBatSpecific;

/* The gsfSeaBatIISpecific data structure is intended to replace the
 * gsfSeaBatSpecific data structure as of GSF_1.04.
 */
typedef struct t_gsfSeaBatIISpecific
{
    int         ping_number;          /* 1 - 32767 */
    double      surface_velocity;      /* meters/second */
    int         mode;                 /* bit mapped, see macros below */
    int         sonar_range;          /* meters */
    int         transmit_power;
    int         receive_gain;
    double      fore_aft_bw;          /* fore/aft beam width in degrees */
    double      athwart_bw;           /* athwartships beam width in degrees */
    char        spare[4];             /* Four bytes of spare space, for future use */
}
t_gsfSeaBatIISpecific;

/* Macro definitions for the SeaBatSpecific and SeaBatIISpecific mode field */
#define GSF_SEABAT_WIDE_MODE      0x01 /* if set 10 deg fore-aft */
#define GSF_SEABAT_9002           0x02 /* if set two sonar heads */
#define GSF_SEABAT_STBD_HEAD      0x04 /* if set starboard ping (seabat head 2) */
#define GSF_SEABAT_9003           0x08 /* if set 9003 series sonar (40 beams) */

/* Define the Reson SeaBat specific data structure */
typedef struct t_gsfSeaBat8101Specific
{
    int         ping_number;          /* 1 - 65535 */
    double      surface_velocity;      /* meters/second */
    int         mode;                 /* bit mapped, see macros below */
    int         range;                /* meters */
    int         power;                /* 0-8 + status bits */
    int         gain;                 /* 1-45 + status bits */
    int         pulse_width;          /* in microseconds */
    int         tvg_spreading;         /* tvg spreading coefficient * 4 */
    int         tvg_absorption;        /* tvg absorption coefficient */
    double      fore_aft_bw;          /* fore/aft beam width in degrees */
    double      athwart_bw;           /* athwartships beam width in degrees */
    double      range_filt_min; /* range filter, minimum value, meters (future use) */
    double      range_filt_max; /* range filter, maximum value, meters (future use) */
    double      depth_filt_min; /* depth filter, minimum value, meters (future use) */
    double      depth_filt_max; /* depth filter, maximum value, meters (future use) */
    int         projector;            /* projector type (future use) */
    char        spare[4];             /* Four bytes of spare space, for future use */
}
t_gsfSeaBat8101Specific;

```

```

/* Macro definitions for the SeaBat8101Specific and SeaBat8101Specific mode field */
#define GSF_8101_WIDE_MODE      0x01    /* set if transmit on receiver */
#define GSF_8101_TWO_HEADS     0x02    /* set if two sonar heads */
#define GSF_8101_STBD_HEAD     0x04    /* set if starboard ping (seabat head 2) */
#define GSF_8101_AMPLITUDE     0x08    /* set if beam amplitude is available (RITHETA
                                         packet) */

/* Define the SeaBeam 2112/36 specific data structure */
typedef struct t_gsfSeaBeam2112Specific
{
    int         mode;                /* bit mapped, see macros below */
    double      surface_velocity;    /* meters/second */
    char        ssv_source;          /* (V)elocimeter, (M)anual, (T)emperature,
                                     (E)xternal, or (U)nknown */

    int         ping_gain;           /* dB */
    int         pulse_width;         /* in milliseconds */
    int         transmitter_attenuation; /* dB */
    int         number_algorithms;   /* algorithms per beam (1-4) */
    char        algorithm_order[5];  /* null terminated string, each char will be
                                     either a space, W(MT), or B(DI). If
                                     number_algorithms equals one, this will be
                                     four spaces */

    char        spare[2];            /* Two bytes of spare space, for future use */
}
t_gsfSeaBeam2112Specific;

/* Macro definitions for the SeaBeam2112Specific mode field */
#define GSF_2112_SVP_CORRECTION 0x01    /* set if true depth, true position corrections
                                         are used */
#define GSF_2112_LOW_FREQUENCY  0x02    /* set if using 12kHz frequency - 36kHz if not
                                         set */
#define GSF_2112_AUTO_DEPTH_GATE 0x04    /* set if depth gate mode is automatic - manual
                                         if not set */

/* SeaBeam 2112 specific macro definitions for the quality factor array */
#define GSF_2112_POOR_QUALITY    0x01    /* set if the beam was flagged by the SeaBeam
                                         as poor quality */
#define GSF_2112_DATA_SOURCE_WMT 0x10    /* set if the data source is WMT - source is
                                         BDI if not set */

/* Define the Elac MkII specific data structure */
typedef struct t_gsfElacMkIISpecific
{
    int         mode;                /* bit mapped, see macros below */
    int         ping_num;
    int         sound_vel;           /* 0.1 m/s */
    int         pulse_length;        /* 0.01 ms */
    int         receiver_gain_stbd;  /* db */
    int         receiver_gain_port;  /* db */
    int         reserved;
}
t_gsfElacMkIISpecific;

/* Macro definitions for the ElacMkIISpecific mode field */
#define GSF_MKII_LOW_FREQUENCY   0x01    /* set if using 12kHz frequency - 36kHz if not
                                         set */
#define GSF_MKII_SOURCE_MODE     0x02    /* set if RDT transmit used, otherwise omni */
#define GSF_MKII_SOURCE_POWER    0x04    /* set if transmit high power - low power if
                                         not set */
#define GSF_MKII_STBD_HEAD       0x08    /* set if starboard ping */

/* Define the Reson SeaBat specific data structure */
typedef struct t_gsfReson7100Specific

```



```

{
    unsigned int    protocol_version;    /* Obtained from the Data Record Frame
                                         (DRF) */
    unsigned int    device_id;           /* i.e. 7101, 7111, 7125, etc. Obtained
                                         from the DRF */
    unsigned char   reserved_1[16];      /* Placeholder for growth of fields from
                                         DRF */
    unsigned int    major_serial_number; /* high order 4 bytes of sonar serial
                                         number, from record 7000 */
    unsigned int    minor_serial_number; /* low order 4 bytes of sonar serial
                                         number, from record 7000 */
    unsigned int    ping_number;         /* sequential number, unique for each
                                         ping, wraps at boundary */
    unsigned int    multi_ping_seq;      /* 0 if not in multi-ping mode, otherwise
                                         number of pings in a multi-ping
                                         sequence */
    double          frequency;           /* Sonar operating frequency in Hz. From
                                         record 7000 */
    double          sample_rate;         /* Sonar system sampling rate in Hz. From
                                         record 7000 */
    double          receiver_bandwidth;  /* Sonar system signal bandwidth in Hz.
                                         From record 7000 */
    double          tx_pulse_width;      /* Transmit pulse length in seconds. From
                                         record 7000 */
    unsigned int    tx_pulse_type_id;    /* 0=CW, 1=Linear chirp, from
                                         record 7000 */
    unsigned int    tx_pulse_envlp_id;   /* 0=Tapered rectangular, 1=Tukey, from
                                         record 7000 */
    unsigned int    tx_pulse_envlp_param; /* four byte field containing envelope
                                         parameter, no definition or units
                                         available, from record 7000 */
    unsigned int    tx_pulse_reserved;   /* four byte field reserved for future
                                         growth, from record 7000 */
    double          max_ping_rate;       /* Maximum ping rate in pings per second,
                                         from record 7000 */
    double          ping_period;         /* seconds since last ping, from
                                         record 7000 */
    double          range;               /* Sonar range selection in meters, from
                                         record 7000 */
    double          power;                /* Power selection in dB re 1 microPa,
                                         from record 7000 */
    double          gain;                 /* Gain selection in dB, from
                                         record 7000 */
    unsigned int    control_flags;       /* 0-3: Auto range method
                                         4-7: Auto bottom detect filter
                                         method
                                         8: Bottom detect range filter
                                         9: Bottom detect depth filter
                                         10-14: Auto receiver gain method
                                         15-31: Reserved */
    unsigned int    projector_id;        /* projector selection, from
                                         record 7000 */
    double          projector_steer_angl_vert; /* degrees, from record 7000 */
    double          projector_steer_angl_horz; /* degrees, from record 7000 */
    double          projector_beam_width_vert; /* degrees, from record 7000 */
    double          projector_beam_width_horz; /* degrees, from record 7000 */
    double          projector_beam_focal_pt;  /* meters, from record 7000 */
    unsigned int    projector_beam_weighting_window_type; /* 0-Rectangular,
                                                             1-Chebyshev,
                                                             from record 7000 */
    unsigned int    projector_beam_weighting_window_param; /* four byte projector
                                                             weighting parameter, no
                                                             definition or units

```

```

                                available, from record
                                7000 */
unsigned int    transmit_flags;    /* 0-3: Pitch stabilization method
                                4-6: Yaw stabilization method
                                8-31: Reserved */
unsigned int    hydrophone_id;    /* hydrophone selection,
                                from record 7000 */
unsigned int    receiving_beam_weighting_window_type; /* 0-Chebyshev, 1-Kaiser,
                                from record 7000 */
unsigned int    receiving_beam_weighting_window_param; /* four byte receiver
                                weighting parameter, no
                                definition or units
                                available, from record
                                7000 */

unsigned int    receive_flags;    /* 0-3: Roll stabilization method
                                4-7: Dynamic focusing method
                                8-11: Doppler compensation method
                                12-15: Match filtering method
                                16-19: TVG method
                                20-23: Multi-Ping Mode
                                24-31: Reserved */

double         receive_beam_width; /* angle in degrees, from record 7000 */
double         range_filt_min;    /* range filter, minimum value, meters,
                                from record 7000 */
double         range_filt_max;    /* range filter, maximum value, meters,
                                from record 7000 */
double         depth_filt_min;    /* depth filter, minimum value, meters,
                                from record 7000 */
double         depth_filt_max;    /* depth filter, maximum value, meters,
                                from record 7000 */
double         absorption;        /* absorption in dB/km, from
                                record 7000 */
double         sound_velocity;    /* sound speed in m/s at transducer, from
                                record 7006 */
double         spreading;        /* spreading loss in dB from
                                record 7000 */
char           reserved_2[16];    /* spare space, for future use */
unsigned char   sv_source;        /* (0: measured, 1: manual), from
                                record 7006 */
unsigned char   layer_comp_flag;  /* (0: off, 1: on), from record 7006 */
char           reserved_3[8];    /* spare space, for future use */
}
t_gsfReson7100Specific;

#define GSF_7100_PITCH_STAB        0x0001 /* set if pitch stabilized */
#define GSF_7100_ROLL_STAB        0x0001 /* set if roll stabilized */

/* Define the Reson 8100 specific data structure */
typedef struct t_gsfReson8100Specific
{
    int          latency;          /* time from ping to output (milliseconds)
*/
    int          ping_number;      /* 4 byte ping number */
    int          sonar_id;         /* least significant 4 bytes of Ethernet
                                address */
    int          sonar_model;      /* */
    int          frequency;        /* KHz */
    double       surface_velocity; /* meters/second */
    int          sample_rate;      /* A/D samples per second */
    int          ping_rate;        /* pings per second * 1000 */
    int          mode;             /* bit mapped, see macros below */
    int          range;            /* meters */
    int          power;            /* 0-8 + status bits */

```

```

        int            gain;                /* 1-45 + status bits */
        int            pulse_width;         /* in microseconds */
        int            tv_g_spreading;      /* tv_g spreading coefficient * 4 */
        int            tv_g_absorption;     /* tv_g absorption coefficient */
        double         fore_aft_bw;         /* fore/aft beam width in degrees */
        double         athwart_bw;          /* athwartships beam width in degrees */
        int            projector_type;       /* projector type */
        int            projector_angle;      /* projector pitch steering angle (degrees *
                                           100) */

        double         range_filt_min;      /* range filter, minimum value, meters */
        double         range_filt_max;      /* range filter, maximum value, meters */
        double         depth_filt_min;      /* depth filter, minimum value, meters */
        double         depth_filt_max;      /* depth filter, maximum value, meters */
        int            filters_active;       /* bit 0 - range filter, bit 1 - depth
                                           filter */
        int            temperature;          /* temperature at sonar head (deg C * 10) */
        double         beam_spacing;         /* across track receive beam angular spacing
                                           */

        char           spare[2];            /* Two bytes of spare space, for future use
    */
}
t_gsfReson8100Specific;

/* Macro definitions for the SeaBat8100Specific mode field */
#define GSF_8100_WIDE_MODE            0x01 /* set if transmit on receiver */
#define GSF_8100_TWO_HEADS            0x02 /* set if two sonar heads */
#define GSF_8100_STBD_HEAD            0x04 /* set if starboard ping (seabat head 2) */
#define GSF_8100_AMPLITUDE            0x08 /* set if beam amplitude is available (RITHETA
packet) */
#define GSF_8100_PITCH_STAB           0x10 /* set if pitch stabilized */
#define GSF_8100_ROLL_STAB            0x20 /* set if roll stabilized */

/* Define the Echotrac Single-Beam sensor specific data structure. */

#define GSF_SB_MPP_SOURCE_UNKNOWN     0x00 /* Unknown MPP source */
#define GSF_SB_MPP_SOURCE_GPS_3S      0x01 /* GPS 3S */
#define GSF_SB_MPP_SOURCE_GPS_TASMAN  0x02 /* GPS Tasman */
#define GSF_SB_MPP_SOURCE_DGPS_TRIMBLE 0x03 /* DGPS Trimble */
#define GSF_SB_MPP_SOURCE_DGPS_TASMAN 0x04 /* DGPS Tasman */
#define GSF_SB_MPP_SOURCE_DGPS_MAG     0x05 /* DGPS MagMPPox */
#define GSF_SB_MPP_SOURCE_RANGE_MFIX   0x06 /* Range/Azimuth - Microfix */
#define GSF_SB_MPP_SOURCE_RANGE_TRIS   0x07 /* Range/Azimuth - Trisponder */
#define GSF_SB_MPP_SOURCE_RANGE_OTHER  0x08 /* Range/Azimuth - Other */

typedef struct t_gsfSBEchoTracSpecific
{
    int            navigation_error;
    unsigned short mpp_source;          /* Flag To determine mpp source - See above */
    unsigned short tide_source;         /* in GSF Version 2.02+ this is in ping flags */
    double         dynamic_draft;       /* speed induced draft in meters */
    char           spare[4];            /* four bytes of reserved space */
}
t_gsfSBEchoTracSpecific;

/* Define the MGD77 Single-Beam sensor specific data structure. */
typedef struct t_gsfSBMGD77Specific
{
    unsigned short time_zone_corr;
    unsigned short position_type_code;
    unsigned short correction_code;
    unsigned short bathy_type_code;
    unsigned short quality_code;

```

```

        double        travel_time;
        char           spare[4];                /* four bytes of reserved space */
    }
    t_gsfSBMGD77Specific;

/* Define the BDB sensor specific data structure */
typedef struct t_gsfSBBDBSpecific
{
    int     doc_no;           /* Document number (5 digits) */
    char    eval;            /* Evaluation (1-best, 4-worst) */
    char    classification; /* Classification ((U)nclass, (C)onfidential,
                                (S)ecret, (P)roprietary/Unclass,
                                (Q)Proprietary/Class) */
    char    track_adj_flag; /* Track Adjustment Flag (Y,N) */
    char    source_flag;    /* Source Flag ((S)urvey, (R)andom, (O)cean Survey) */
    char    pt_or_track_ln; /* Discrete Point (D) or Track Line (T) Flag */
    char    datum_flag;     /* Datum Flag ((W)GS84, (D)atumless) */
    char    spare[4];       /* four bytes of reserved space */
}
t_gsfSBBDBSpecific;

/* Define the NOS HDB sensor specific data structure */
typedef struct t_gsfSBNOSHDBSpecific
{
    unsigned short type_code; /* Depth type code */
    unsigned short carto_code; /* Cartographic code */
    char           spare[4]; /* four bytes of reserved space */
}
t_gsfSBNOSHDBSpecific;

/* Define the Navisound sensor specific data structure */
typedef struct t_gsfSBNavisoundSpecific
{
    double        pulse_length; /* pulse length in cm */
    char           spare[8];     /* eight bytes of reserved space */
}
t_gsfSBNavisoundSpecific;

/* Define the GeoSwath sensor specific data structure */
typedef struct t_gsfGeoSwathPlusSpecific
{
    int     data_source; /* 0 = CBF, 1 = RDF */
    int     side;        /* 0 = port, 1 = starboard */
    int     model_number; /* ie: 100, 250, 500, ... */
    double  frequency;   /* Hz */
    int     echosounder_type; /* ? */
    long    ping_number; /* 0 - 4,294,967,295 */
    int     num_nav_samples; /* number of navigation samples in this ping */
    int     num_attitude_samples; /* number of attitude samples in this ping */
    int     num_heading_samples; /* number of heading samples in this ping */
    int     num_minisVS_samples; /* number of minisVS samples in this ping */
    int     num_echosounder_samples; /* number of echosounder samples in ping */
    int     num_raa_samples; /* number of RAA (Range/Angle/Amplitude) samples in ping */
    double  mean_sv; /* meters per second */
    double  surface_velocity; /* in m/s */
    int     valid_beams; /* number of valid beams for this ping */
    double  sample_rate; /* Hz */
    double  pulse_length; /* micro seconds */
}

```

```

        int            ping_length;                /* meters */
        int            transmit_power;            /* ? */
        int            sidescan_gain_channel;      /* RDF documentation = 0 - 3 */
        int            stabilization;              /* 0 or 1 */
        int            gps_quality;               /* ? */
        double         range_uncertainty;          /* meters */
        double         angle_uncertainty;          /* degrees */
        char            spare[32];                 /* 32 bytes of reserved space */
    }
    t_gsfGeoSwathPlusSpecific;

#define GSF_GEOSWATH_PLUS_PORT_PING 0
#define GSF_GEOSWATH_PLUS_STBD_PING 1

/* Macro definitions for EM4 series sector data details */
#define GSF_MAX_EM4_SECTORS 9
/* Macro definitions for EM3 series sector data details */
#define GSF_MAX_EM3_SECTORS 20

/* Define sub-structure for the transmit sectors */
#define GSF_EM_WAVEFORM_CW 0
#define GSF_EM_WAVEFORM_FM_UP 1
#define GSF_EM_WAVEFORM_FM_DOWN 2

typedef struct t_gsfEM4TxSector
{
    double            tilt_angle;                  /* transmitter tilt angle in degrees */
    double            focus_range;                 /* focusing range, 0.0 for no focusing */
    double            signal_length;               /* transmit signal duration in seconds */
    double            transmit_delay;              /* Sector transmit delay from first
    transmission in seconds */
    double            center_frequency;            /* center frequency in Hz */
    double            mean_absorption;             /* mean absorption coefficient in 0.01
    dB/kilometer */
    int               waveform_id;                 /* signal waveform ID 0=CW; 1=FM upsweep;
    2=FM downsweep */
    int               sector_number;               /* transmit sector number */
    double            signal_bandwidth;            /* signal bandwidth in Hz */
    unsigned char     spare[16];                  /* spare space */
}
t_gsfEM4TxSector;

typedef struct t_gsfEM3RawTxSector
{
    double            tilt_angle;                  /* transmitter tilt angle in degrees */
    double            focus_range;                 /* focusing range, 0.0 for no focusing */
    double            signal_length;               /* transmit signal duration in seconds */
    double            transmit_delay;              /* Sector transmit delay from first
    transmission in seconds */
    double            center_frequency;            /* center frequency in Hz */
    int               waveform_id;                 /* signal waveform ID 0=CW; 1=FM upsweep;
    2=FM downsweep */
    int               sector_number;               /* transmit sector number */
    double            signal_bandwidth;            /* signal bandwidth in Hz */
    unsigned char     spare[16];                  /* spare space */
}
t_gsfEM3RawTxSector;

/* The following macro definitions are to aid in interpretation of the sonar mode field */
/*
#define GSF_EM_MODE_VERY_SHALLOW 0x00          /* Bits 2,1,0 cleared means very shallow
mode */

```

```

#define GSF_EM_MODE_SHALLOW      0x01      /* Bit zero set means shallow mode */
#define GSF_EM_MODE_MEDIUM        0x02      /* Bit one set means medium mode */
#define GSF_EM_MODE_DEEP          0x03      /* Bits one and zero set means deep
mode */
#define GSF_EM_MODE_VERY_DEEP     0x04      /* Bit two set means very deep mode */
#define GSF_EM_MODE_EXTRA_DEEP    0x05      /* Bits two and one set means extra deep
mode */
#define GSF_EM_MODE_MASK          0x07      /* Mask off bits 2,1,0 to determine just
the mode */
/* Exact definition of bits 5,4,3 not
clear from document rev J. */
#define GSF_EM_MODE_DS_OFF        0xC0      /* bits 7 and 6 cleared means dual swath
off */
#define GSF_EM_MODE_DS_FIXED      0x40      /* bit 6 set means dual swath in fixed
mode */
#define GSF_EM_MODE_DS_DYNAMIC    0x80      /* bit 7 set means dual swath in dynamic
mode */

/* Define a data structure to hold the Simrad EM series run time parameters per datagram
document rev I. */
typedef struct t_gsfEMRunTime
{
    int          model_number;      /* from the run-time parameter datagram
*/
    struct timespec dg_time;      /* from the run-time parameter datagram
*/
    int          ping_counter;      /* sequential counter 0 - 65535 */
    int          serial_number;     /* The primary sonar head serial number
*/
    unsigned char operator_station_status; /* Bit mask of status information for
operator station */
    unsigned char processing_unit_status; /* Bit mask of status information for
sonar processor unit */
    unsigned char bsp_status;      /* Bit mask of status information for BSP
status */
    unsigned char head_transceiver_status; /* Bit mask of status information for
sonar head or sonar transceiver */
    unsigned char mode;            /* Bit mask of sonar operating
information, see mode bit mask
definitions */
    unsigned char filter_id;      /* one byte bit mask for various sonar
processing filter settings */
    double        min_depth;      /* meters */
    double        max_depth;      /* meters */
    double        absorption;     /* dB/km */
    double        tx_pulse_length; /* in micro seconds */
    double        tx_beam_width;  /* degrees */
    double        tx_power_re_max; /* The transmit power referenced to
maximum power in dB */
    double        rx_beam_width;  /* degrees */
    double        rx_bandwidth;   /* Hz */
    double        rx_fixed_gain;  /* dB */
    double        tvg_cross_over_angle; /* degrees */
    unsigned char ssv_source;     /* one byte bit mask defining SSSV source
-> 0=sensor, 1=manual, 2=profile */
    int          max_port_swath_width; /* total swath width to port side in
meters */
    unsigned char beam_spacing;   /* one byte bit mask -> 0=beamwidth,
1=equiangle, 2=equidistant,
3=intermediate */
    int          max_port_coverage; /* coverage to port side in degrees */
    unsigned char stabilization; /* one byte bit mask defining yaw and
pitch stabilization mode */

```

```

    int            max_stbd_coverage;        /* coverage to starboard side in degrees
*/
    int            max_stbd_swath_width;      /* total swath width to starboard side in
                                                meters */
    double         durotong_speed;           /* Sound speed in durotong for the EM1002
                                                transducer, zero if not available */
    double         hi_low_absorption_ratio;   /* Absorption coeffieceint ratio */
    double         tx_along_tilt;             /* Transmit fan along track tilt angle in
                                                degrees */
    unsigned char   filter_id_2;              /* two lowest order bits define the
                                                penetration filter setting: off, weak,
                                                medium, or strong */
    unsigned char   spare[16];               /* 16 spare bytes */
}
t_gsfEMRunTime;

/* Macro definitions for bits of pu_status field */
#define GSF_EM_VALID_1_PPS      0x0001      /* If set, then 1 PPS timing is valid */
#define GSF_EM_VALID_POSITION  0x0002      /* If set, then position input is valid */
#define GSF_EM_VALID_ATTITUDE  0x0004      /* If set, then attitude input is valid */
#define GSF_EM_VALID_CLOCK     0x0008      /* If set, then clock status is valid */
#define GSF_EM_VALID_HEADING   0x0010      /* If set, then heading status is valid */
#define GSF_EM_PU_ACTIVE       0x0020      /* If set, then PU is active (i.e.
                                                pinging) */

/* Define a data structure to hold the Simrad EM series PU status values per datagram
document rev I. */
typedef struct t_gsfEMPUSatus
{
    double         pu_cpu_load;              /* Percent CPU load in the processor unit
*/
    unsigned short sensor_status;            /* Bit mask containing status of sensor
inputs */
    int            achieved_port_coverage;    /* Achieved coverage to port in degrees */
    int            achieved_stbd_coverage;    /* Achieved coverage to starboard in
degrees */
    double         yaw_stabilization;         /* in degrees */
    unsigned char   spare[16];
}
t_gsfEMPUSatus;

/* Define sensor specific data structures for the Kongsberg 710/302/122 */
typedef struct t_gsfEM4Specific
{
    /* values from the XYZ datagram and raw range datagram */
    int            model_number;              /* 122, or 302, or 710, or ... */
    int            ping_counter;              /* Sequential ping counter, 1 through
                                                65535 */
    int            serial_number;             /* System unique serial number, 100 - ? */
    double         surface_velocity;          /* Measured sound speed near the surface
                                                in m/s */
    double         transducer_depth;          /* The transmit transducer depth in meters
re water level at ping time */
    int            valid_detections;          /* number of beams with a valid bottom
detection for this ping */
    double         sampling_frequency;        /* The system digitizing rate in Hz */
    unsigned int   doppler_corr_scale;        /* Scale factor value to be applied to
Doppler correction field prior to
applying corrections */
    double         vehicle_depth;             /* From 0x66 datagram, non-zero when
sonar head is mounted on a sub-sea
platform */
    unsigned char   spare_1[16];

```

```

    int            transmit_sectors;          /* The number of transmit sectors for
                                                this ping */
    t_gsfEM4TxSector sector[GSF_MAX_EM4_SECTORS]; /* Array of structures with transmit
                                                sector information */
    unsigned char   spare_2[16];

    /* Values from the run-time parameters datagram */
    t_gsfEMRunTime  run_time;

    /* Values from the PU status datagram */
    t_gsfEMPUSStatus pu_status;
}
t_gsfEM4Specific;

/* Define sensor specific data structures for the Kongsberg 3000, etc which use raw
range and beam angle */
typedef struct t_gsfEM3RawSpecific
{
    /* values from the XYZ datagram and raw range datagram */
    int            model_number;              /* ie 3000 ... */
    int            ping_counter;              /* Sequential ping counter, 0 through
                                                65535 */
    int            serial_number;             /* System unique serial number,
                                                100 - ? */
    double         surface_velocity;           /* Measured sound speed near the surface
                                                in m/s */
    double         transducer_depth;          /* The transmit transducer depth in
                                                meters re water level at ping time */
    int            valid_detections;          /* number of beams with a valid bottom
                                                detection for this ping */
    double         sampling_frequency;        /* The system digitizing rate in Hz */
    double         vehicle_depth;             /* vechicle depth in 0.01 m */
    double         depth_difference;          /* in meters between sonar heads in
                                                em3000d configuration */
    int            offset_multiplier;         /* transducer depth offset multiplier */
    unsigned char   spare_1[16];
    int            transmit_sectors;          /* The number of transmit sectors for
                                                this ping */
    t_gsfEM3RawTxSector sector[GSF_MAX_EM3_SECTORS]; /* Array of structures with
                                                transmit sector information */
    unsigned char   spare_2[16];

    /* Values from the run-time parameters datagram */
    t_gsfEMRunTime  run_time;

    /* Values from the PU status datagram */
    t_gsfEMPUSStatus pu_status;
}
t_gsfEM3RawSpecific;

/* Define the Klein 5410 Bathy Sidescan sensor specific data structure */
typedef struct t_gsfKlein5410BssSpecific
{
    int            data_source;               /* 0 = SDF */
    int            side;                     /* 0 = port, 1 = stbd */
    int            model_number;              /* ie: 5410 */
    double         acoustic_frequency;        /* system frequency in Hz */
    double         sampling_frequency;        /* sampling frequency in Hz */
    unsigned int    ping_number;              /* 0 - 4,294,967,295 */
    unsigned int    num_samples;              /* total number of samples in this ping */
    unsigned int    num_raa_samples;          /* number of valid range, angle, amplitude
samples in ping */

```



```

        unsigned int    error_flags;           /* error flags for this ping */
        unsigned int    range;                 /* sonar range setting */
        double          fish_depth;            /* reading from the towfish pressure sensor
in Volts */
        double          fish_altitude;         /* towfish altitude in m */
        double          sound_speed;           /* speed of sound at the transducer face in
m/sec */
        int             tx_waveform;           /* transmit pulse: 0 = 132 microsec CW; 1 =
132 microsec FM; */
                                                /* 2 = 176 microsec CW; 3 = 176 microsec FM
*/
        int             altimeter;             /* altimeter status: 0 = passive, 1 =
active */
        unsigned int    raw_data_config;       /* raw data configuration */
        char            spare[32];             /* 32 bytes of reserved space */
    }
    t_gsfKlein5410BssSpecific;

/* Define the Imagenex Delta T sensor specific data structure */
typedef struct t_gsfDeltaTSpecific
{
    char                decode_file_type[4];    /* contains the decoded files extension. */
    char                version;               /* contains the minor version number of the
delta t */
    int                 ping_byte_size;        /* size in bytes of this ping (256 +
(((byte 117[1 or 0])*2) + 2) * number of beams)) */
    struct timespec     interrogation_time;     /* The sonar interrogation time */
    int                 samples_per_beam;      /* number of samples per beam */
    double              sector_size;           /* size of the sector in degrees */
    double              start_angle;           /* the angle that beam 0 starts at in
degrees. */
    double              angle_increment;        /* the number of degrees the angle
increments per beam */
    int                 acoustic_range;         /* acoustic range in meters */
    int                 acoustic_frequency;     /* acoustic frequency in kHz */
    double              sound_velocity;         /* the velocity of sound at the transducer
face in m/s */
    double              range_resolution;       /* range resolution in centimeters
(documentation says mm but all example data is in cm) */
    double              profile_tilt_angle;     /* the mounting offset */
    double              repetition_rate;        /* time between pings in milliseconds */
    unsigned long       ping_number;           /* the current ping number of this ping.
*/
    unsigned char       intensity_flag;         /* this tells whether the GSF will have
intensity data (1=true) */
    double              ping_latency;          /* time from sonar ping interrogation to
actual ping in seconds */
    double              data_latency;          /* time from sonar ping interrogation to
83P UDP datagram in seconds */
    unsigned char       sample_rate_flag;      /* sampling rate 0 = (1 in 500); 1 = (1 in
5000) */
    unsigned char       option_flags;          /* this flag states whether the data is
roll corrected or raybend corrected (1 = roll, 2 = raybend, 3 = both) */
    int                 num_pings_avg;         /* number of pings averaged 1 - 25 */
    double              center_ping_time_offset; /* the time difference in seconds between
the center ping interrogation and the current ping interrogation */
    unsigned char       user_defined_byte;     /* contains a user defined byte */
    double              altitude;              /* the height of the fish above the ocean
floor. */
    char                external_sensor_flags; /* this flag is a bit mask where (1 =
external heading, 2 = external roll, 4 = external pitch, 8 = external heave) */
    double              pulse_length;          /* acoustic pulse length in seconds */

```

```

        double          fore_aft_beamwidth;      /* Effective f/a beam width in degrees */
        double          athwartships_beamwidth; /* Effective athwartships beam width in
degrees */
        unsigned char    spare[32];              /* room to grow */
    }
    t_gsfDeltaTSpecific;

/* Define a union of the known sensor specific ping subrecords */
typedef union t_gsfSensorSpecific
{
    t_gsfSeaBeamSpecific      gsfSeaBeamSpecific;
    t_gsfEM100Specific        gsfEM100Specific;
    t_gsfEM121ASpecific       gsfEM121ASpecific;
    t_gsfEM121ASpecific       gsfEM121Specific;
    t_gsfSeaBatSpecific       gsfSeaBatSpecific;
    t_gsfEM950Specific        gsfEM950Specific;
    t_gsfEM950Specific        gsfEM1000Specific;
    t_gsfSeamapSpecific       gsfSeamapSpecific;
    /*
    * The following two subrecords are expected to be replaced
    * in a future release by the gsfCmpSassSpecific subrecord.
    */
    t_gsfTypeIIISpecific      gsfTypeIIISeaBeamSpecific;
    t_gsfTypeIIISpecific      gsfSASSSpecific;

    t_gsfCmpSassSpecific      gsfCmpSassSpecific;
    t_gsfSBampSpecific        gsfSBampSpecific;
    t_gsfSeaBatIISpecific     gsfSeaBatIISpecific;
    t_gsfSeaBat8101Specific   gsfSeaBat8101Specific;
    t_gsfSeaBeam2112Specific   gsfSeaBeam2112Specific;
    t_gsfElacMkIISpecific     gsfElacMkIISpecific;
    t_gsfEM3Specific          gsfEM3Specific;
    t_gsfEM3RawSpecific        gsfEM3RawSpecific;
    t_gsfReson7100Specific     gsfReson7100Specific;
    t_gsfReson8100Specific     gsfReson8100Specific;
    t_gsfGeoSwathPlusSpecific  gsfGeoSwathPlusSpecific;
    t_gsfEM4Specific          gsfEM4Specific;
    t_gsfKlein5410BssSpecific gsfKlein5410BssSpecific;

    /* Single beam sensors added */
    t_gsfSBechotracSpecific    gsfSBechotracSpecific;
    t_gsfSBechotracSpecific    gsfSBBathy2000Specific;
    t_gsfSBMGD77Specific       gsfSBMGD77Specific;
    t_gsfSBBDBSpecific         gsfSBBDBSpecific;
    t_gsfSBNOSHDBSpecific      gsfSBNOSHDBSpecific;
    t_gsfSBechotracSpecific    gsfSBPDDSpecific;
} gsfSensorSpecific;

```

Table 4-1 Sensor ID allocation to Sensor Specific Subrecord Data Structure

Sensor ID	Sensor Specific Subrecord Structure
GSF_SWATH_BATHY_SUBRECORD_SEABEAM_SPECIFIC	gsfSeaBeamSpecific
GSF_SWATH_BATHY_SUBRECORD_EM100_SPECIFIC	gsfEM100Specific
GSF_SWATH_BATHY_SUBRECORD_EM121A_SPECIFIC	gsfEM121ASpecific
GSF_SWATH_BATHY_SUBRECORD_EM121_SPECIFIC	gsfEM121Specific
GSF_SWATH_BATHY_SUBRECORD_SEABAT_SPECIFIC	gsfSeaBatSpecific
GSF_SWATH_BATHY_SUBRECORD_EM950_SPECIFIC	gsfEM950Specific
GSF_SWATH_BATHY_SUBRECORD_EM1000_SPECIFIC	gsfEM1000Specific
GSF_SWATH_BATHY_SUBRECORD_SEAMAP_SPECIFIC	gsfSeamapSpecific

GSF_SWATH_BATHY_SUBRECORD_TYPEIII_SEABEAM_SPECIFIC	gsfTypeIIISeaBeamSpecific
GSF_SWATH_BATHY_SUBRECORD_SASS_SPECIFIC	gsfSASSSpecific
GSF_SWATH_BATHY_SUBRECORD_CMP_SASS_SPECIFIC	gsfCmpSassSpecific
GSF_SWATH_BATHY_SUBRECORD_SB_AMP_SPECIFIC	gsfSBampSpecific
GSF_SWATH_BATHY_SUBRECORD_SEABAT_II_SPECIFIC	gsfSeaBatIISpecific
GSF_SWATH_BATHY_SUBRECORD_SEABAT_8101_SPECIFIC	gsfSeaBat8101Specific
GSF_SWATH_BATHY_SUBRECORD_SEABEAM_2112_SPECIFIC	gsfSeaBeam2112Specific
GSF_SWATH_BATHY_SUBRECORD_ELAC_MKII_SPECIFIC	gsfElacMkIISpecific
GSF_SWATH_BATHY_SUBRECORD_EM3000_SPECIFIC	gsfEM3Specific
GSF_SWATH_BATHY_SUBRECORD_EM1002_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM300_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM120_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3002_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3000D_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3002D_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM121A_SIS_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM2000_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_RESON_8101_SPECIFIC	gsfReson8100Specific
GSF_SWATH_BATHY_SUBRECORD_RESON_8111_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_RESON_8124_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_RESON_8125_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_RESON_8150_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_RESON_8160_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_GEOSWATH_PLUS_SPECIFIC	gsfGeoSwathPlusSpecific
GSF_SWATH_BATHY_SUBRECORD_EM710_SPECIFIC	gsfEM4Specific
GSF_SWATH_BATHY_SUBRECORD_EM302_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM122_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM2040_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_KLEIN_5410_BSS_SPECIFIC	gsfKlein5410BssSpecific
GSF_SWATH_BATHY_SUBRECORD_RESON_7125_SPECIFIC	gsfReson7100Specific
GSF_SWATH_BATHY_SUBRECORD_EM300_RAW_SPECIFIC	gsfEM3RawSpecific
GSF_SWATH_BATHY_SUBRECORD_EM1002_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM2000_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3000_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM120_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3002_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3000D_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM3002D_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_EM121A_SIS_RAW_SPECIFIC	
GSF_SWATH_BATHY_SUBRECORD_DELTA_T_SPECIFIC	gsfDeltaTSpecific

4.1.2.3 Bathymetric Receive Beam Time Series Intensity Subrecord

```
typedef struct gsftimeSeriesIntensity
{
    unsigned short sample_count;      /* number of amplitude samples Per beam */
    unsigned short detect_sample;     /* index of bottom detection sample for the beam */
    unsigned char  spare[8];          /* for future use */
    unsigned int   *samples;           /* Array of per-beam time series intensity samples
                                         */
} gsftimeSeriesIntensity;

#define GSF_INTENSITY_LINEAR      (unsigned)0x01
#define GSF_INTENSITY_CALIBRATED (unsigned)0x02
#define GSF_INTENSITY_POWER      (unsigned)0x04
#define GSF_INTENSITY_GAIN       (unsigned)0x08

typedef struct t_gsfbRBIntensity
{
    unsigned char      bits_per_sample; /* bits per intensity sample */
    unsigned int        applied_corrections; /* flags to describe corrections
                                         applied to intensity values */
}
```

```

    unsigned char        spare[16];                /* spare header space */
    gsfsensorimagery     sensor_imagery;           /* sensor specific per-ping imagery
                                                    information */
    gsfsTimeSeriesIntensity *time_series;          /* array of per-beam time series
                                                    intensity records */
} gsfsBRBIntensity;

typedef struct t_gsfsEM3ImagerySpecific
{
    unsigned short range_norm;                     /* range to normal incidence used to correct
                                                    sample amplitudes (in samples) */
    unsigned short start_tvg_ramp;                 /* start range sample of TVG ramp if not enough
                                                    dynamic range (0 else) */
    unsigned short stop_tvg_ramp;                 /* stop range sample of TVG ramp if not enough
                                                    dynamic range (0 else) */
    char               bsn;                       /* normal incidence BS in dB */
    char               bso;                       /* oblique BS in dB */
    double             mean_absorption;           /* mean absorption coefficient in dB/km,
                                                    resolution of 0.01 dB/km */
    short              offset;                   /* Value that has been added to all imagery
                                                    samples to convert to a positive value */
    short              scale;                   /* Manufacturer's specified scale value for each
                                                    sample. This value is 2 for data from
                                                    EM3000EM3002/EM1002/EM300/EM120 */
    unsigned char      spare[4];                 /* spare sensor specific subrecord space,
                                                    reserved for future expansion */
} t_gsfsEM3ImagerySpecific;

typedef struct t_gsfsReson7100ImagerySpecific
{
    unsigned short size;
    unsigned char      spare[64];                 /* spare sensor specific subrecord space,
                                                    reserved for future expansion */
} t_gsfsReson7100ImagerySpecific;

typedef struct t_gsfsReson8100ImagerySpecific
{
    unsigned char      spare[8];                 /* spare sensor specific subrecord space,
                                                    reserved for future expansion */
} t_gsfsReson8100ImagerySpecific;

typedef struct t_gsfsEM4ImagerySpecific
{
    double             sampling_frequency;        /* The system digitizing rate in Hz, value
                                                    retrieved from the imagery datagram */
    double             mean_absorption;           /* mean absorption coefficient in dB/km, from
                                                    0x53 datagram, 0 if data is from 0x59 */
    double             tx_pulse_length;          /* transmit pulse length in microseconds from
                                                    imagery datagram 0x53, or 0x59 */
    int                range_norm;               /* range to normal incidence used to correct
                                                    sample amplitudes (in samples) */
    int                start_tvg_ramp;           /* start range (in samples) of TVG ramp if not
                                                    enough dynamic range 0 means not used
*/
    int                stop_tvg_ramp;           /* stop range (in samples) of TVG ramp if not
                                                    enough dynamic range 0 means not used */
    double             bsn;                     /* normal incidence BS in dB */
    double             bso;                     /* oblique incidence BS in dB */
    double             tx_beam_width;           /* transmit beam width in degrees from imagery
                                                    datagram */
    double             tvg_cross_over;          /* The TVG law crossover angle in degrees */
    short              offset;                   /* Value that has been added to all imagery
                                                    samples to convert to a positive value */

```

```

        short            scale;                /* Manufacturer's specified scale value for each
                                                sample. This value is 10 for data from
                                                EM710/EM302/EM122 */
        unsigned char    spare[20];           /* spare sensor specific subrecord space,
                                                reserved for future expansion */
    } t_gsfEM4ImagerySpecific;

typedef struct t_gsfKlein5410BssImagerySpecific
{
    unsigned int    res_mode;                /* Descriptor for resolution mode: 0 = normal; 1
= high */
    unsigned int    tvg_page;                /* TVG page number */
    unsigned int    beam_id[5];              /* array of identifiers for five sidescan beam
magnitude time series, starting with beam id 1 as the forward-most */
    unsigned char    spare[4];                /* spare sensor specific subrecord space,
reserved for future expansion */
} t_gsfKlein5410BssImagerySpecific;

typedef union t_gsfSensorImagery
{
    t_gsfEM3ImagerySpecific    gsfEM3ImagerySpecific;    /* used for EM120,
                                                            EM300, EM1002, EM3000,
                                                            EM3002, and EM121A_SIS */
    t_gsfReson7100ImagerySpecific    gsfReson7100ImagerySpecific; /* For Reson 71P
                                                            "snippet" imagery */
    t_gsfReson8100ImagerySpecific    gsfReson8100ImagerySpecific; /* For Reson 81P
                                                            "snippet" imagery */
    t_gsfEM4ImagerySpecific    gsfEM4ImagerySpecific;    /* used for EM122,
                                                            EM302, EM710 */
    t_gsfKlein5410BssImagerySpecific    gsfKlein5410BssImagerySpecific; /* used for Klein
                                                            5410 Bathy
                                                            Sidescan */
} gsfSensorImagery;

```

4.1.3 Single-beam Bathymetry Record

```

/* Define a single beam record structure */
typedef struct t_gsfSingleBeamPing
{
    struct timespec ping_time;                /* Time the sounding was made */
    double    latitude;                      /* latitude (degrees) of sounding */
    double    longitude;                    /* longitude (degrees) of sounding */
    double    tide_corrector;                /* in meters */
    double    depth_corrector;              /* in meters, draft corrector for sensor */
    double    heading;                      /* in degrees */
    double    pitch;                        /* in meters */
    double    roll;                         /* in meters */
    double    heave;                       /* in meters */
    double    depth;                        /* in meters */
    double    sound_speed_correction;        /* in meters */
    unsigned short    positioning_system_type;
    int    sensor_id;
    gsfSBSensorSpecific    sensor_data;
}
gsfSingleBeamPing;

```

Note that while GSF maintains both read and write support for the Single-Beam record definition, users are actively discouraged from using this record. The preferred means of saving single beam data is to use the `gsfSwathBathyPing` record definition, with the number `beams` field set to one.

4.1.3.1 Single-beam Sensor-specific Subrecords

```
/* Define the Echotrac Single-Beam sensor specific data structure. */
typedef struct t_gsfEchotracSpecific
{
    int                navigation_error;
    unsigned short     mpp_source;      /* Flag To determine if nav was mpp */
    unsigned short     tide_source;
}
t_gsfEchotracSpecific;

/* Define the MGD77 Single-Beam sensor specific data structure. */
typedef struct t_gsfMGD77Specific
{
    unsigned short     time_zone_corr;
    unsigned short     position_type_code;
    unsigned short     correction_code;
    unsigned short     bathy_type_code;
    unsigned short     quality_code;
    double             travel_time;
}
t_gsfMGD77Specific;

/* Define the BDB sensor specific data structure */
typedef struct t_gsfBDBSpecific
{
    int    doc_no;          /* Document number (5 digits) */
    char   eval;           /* Evaluation (1-best, 4-worst) */
    char   classification; /* Classification ((U)nclass, (C)onfidential,
                           (S)ecret, (P)roprietary/Unclass,
                           (Q)Proprietary/Class) */
    char   track_adj_flag; /* Track Adjustment Flag (Y,N) */
    char   source_flag;    /* Source Flag ((S)urvey, (R)andom, (O)cean Survey) */
    char   pt_or_track_ln; /* Discrete Point (D) or Track Line (T) Flag */
    char   datum_flag;     /* Datum Flag ((W)GS84, (D)atumless) */
}
t_gsfBDBSpecific;

/* Define the NOS HDB sensor specific data structure */
typedef struct t_gsfNOSHDBSpecific
{
    unsigned short     type_code;      /* Depth type code */
    unsigned short     carto_code;     /* Cartographic code */
}
t_gsfNOSHDBSpecific;
```

4.1.4 Sound Velocity Profile (SVP) Record

```
typedef struct t_gsfSVP
{
    struct timespec observation_time; /* time the SVP measurement was made */
    struct timespec application_time; /* time the SVP was used by the sonar */
    double          latitude;         /* latitude (degrees) of SVP measurement */
    double          longitude;        /* longitude (degrees) of SVP measurement */
    int             number_points;    /* number of data points in the profile */
    double          *depth;           /* array of profile depth values in meters */
    double          *sound_speed;     /* array of profile sound velocity values in m/s */
}
```

```

}
gsfSVP;

```

4.1.5 Processing Parameters Record

```

#define GSF_MAX_PROCESSING_PARAMETERS 128
typedef struct t_gsfProcessingParameters
{
    struct timespec param_time;
    int    number_parameters;
    short  param_size[GSF_MAX_PROCESSING_PARAMETERS]; /* array of sizes of param text*/
    char   *param[GSF_MAX_PROCESSING_PARAMETERS];     /* array of parameters:
                                                         "param_name=param_value" */
}
gsfProcessingParameters;

```

4.1.5.1 Internal Structure for Processing Parameters

```

#define GSF_MAX_OFFSETS          2
#define GSF_COMPENSATED          1
#define GSF_UNCOMPENSATED        0
#define GSF_TRUE_DEPTHS         1
#define GSF_DEPTHS_RE_1500_MS    2
#define GSF_DEPTH_CALC_UNKNOWN   3
#define GSF_UNKNOWN_PARAM_VALUE  DBL_MIN /* defined in <float.h> */
#define GSF_TRUE                 1
#define GSF_FALSE                0

/* Macro definitions for type of platform */
#define GSF_PLATFORM_TYPE_SURFACE_SHIP 0 /* Add for AUV vs Surface Ship
                                         discrimination */
#define GSF_PLATFORM_TYPE_AUV         1 /* Add for AUV vs Surface Ship
                                         discrimination */
#define GSF_PLATFORM_TYPE_ROTVP       2

typedef struct t_gsfMBOffsets
{
    double  draft[GSF_MAX_OFFSETS];          /* meters */
    double  roll_bias[GSF_MAX_OFFSETS];       /* degrees */
    double  pitch_bias[GSF_MAX_OFFSETS];      /* degrees */
    double  gyro_bias[GSF_MAX_OFFSETS];       /* degrees */
    double  position_x_offset;                /* meters */
    double  position_y_offset;                /* meters */
    double  position_z_offset;                /* meters */
    double  antenna_x_offset;                 /* meters */
    double  antenna_y_offset;                 /* meters */
    double  antenna_z_offset;                 /* meters */
    double  transducer_x_offset[GSF_MAX_OFFSETS]; /* meters */
    double  transducer_y_offset[GSF_MAX_OFFSETS]; /* meters */
    double  transducer_z_offset[GSF_MAX_OFFSETS]; /* meters */
    double  transducer_pitch_offset[GSF_MAX_OFFSETS]; /* degrees */
    double  transducer_roll_offset[GSF_MAX_OFFSETS]; /* degrees */
    double  transducer_heading_offset[GSF_MAX_OFFSETS]; /* degrees */
    double  mru_roll_bias;                    /* degrees */
    double  mru_pitch_bias;                   /* degrees */
    double  mru_heading_bias;                 /* degrees */
    double  mru_x_offset;                     /* meters */
    double  mru_y_offset;                     /* meters */
    double  mru_z_offset;                     /* meters */
}

```

```

    double    position_latency;                /* seconds */
    double    attitude_latency;               /* seconds */
    double    depth_sensor_latency;           /* seconds */
} gsfMBOffsets;

/* Define a data structure to hold multibeam sonar processing parameters */
typedef struct t_gsfMBParams
{
    /* These parameters define reference points */
    char start_of_epoch[64];
    int horizontal_datum;
    int vertical_datum;

    /* These parameters specify what corrections have been applied to the data */
    int roll_compensated;    /* = GSF_COMPENSATED if depth data roll corrected */
    int pitch_compensated;  /* = GSF_COMPENSATED if depth data pitch corrected */
    int heave_compensated;  /* = GSF_COMPENSATED if depth data heave corrected */
    int tide_compensated;   /* = GSF_COMPENSATED if depth data tide corrected */
    int ray_tracing;        /* = GSF_COMPENSATED if travel time/angle pairs are
                             compensated for ray tracing */
    int depth_calculation;  /* = GSF_TRUE_DEPTHS, or GSF_DEPTHS_RE_1500_MS,
                             applicable to the depth field */
    int vessel_type;        /* Surface ship, AUV, etc. */
    int full_raw_data;      /* = GSF_TRUE all data required for full
                             recalculation */
    int msb_applied_to_attitude; /* = GSF_TRUE if contains motion sensor biases */
    int heave_removed_from_gps_tc; /* = GSF_TRUE if heave removed from
                                     gps_tide_corrector */

    /* These parameters specify known offsets that have NOT been corrected.
     * If each of these values are zero, then all known offsets have been
     * corrected for.
     */
    gsfMBOffsets to_apply;

    /* These parameters specify offsets which have already been corrected. */
    gsfMBOffsets applied;
} gsfMBParams;

```

4.1.6 Sensor Parameters Record

```

#define GSF_MAX_SENSOR_PARAMETERS 128
typedef struct t_gsfSensorParameters
{
    struct timespec param_time;
    int    number_parameters;
    short   param_size[GSF_MAX_SENSOR_PARAMETERS]; /* array of sizes of param text */
    char    *param[GSF_MAX_SENSOR_PARAMETERS];      /* array of parameters:
                                                         "param_name=param_value" */
}
gsfSensorParameters;

```

4.1.7 Comment Record

```

typedef struct t_gsfComment
{
    struct timespec comment_time;
    int    comment_length;
    char    *comment;
}

```



```

}
gsfComment;

```

4.1.8 History Record

```

#define GSF_OPERATOR_LENGTH 64
#define GSF_HOST_NAME_LENGTH 64
typedef struct t_gsfHistory
{
    struct timespec history_time;
    char             host_name[GSF_HOST_NAME_LENGTH + 1];
    char             operator_name[GSF_OPERATOR_LENGTH + 1];
    char             *command_line;
    char             *comment;
}
gsfHistory;

```

4.1.9 Navigation Error Record

Note: As of GSF v1.07, the *gsfNavigationError* record has been replaced by *gsfHVNavigationError*. All newly created files should be written using *gsfHVNavigationError*, instead of *gsfNavigationError*.

```

typedef struct t_gsfNavigationError /* obsolete, as of GSF v1.07 */
{
    struct timespec nav_error_time;
    int             record_id;          /* Containing nav with these errors */
    double          latitude_error;     /* 90% CE in meters */
    double          longitude_error;    /* 90% CE in meters */
}
gsfNavigationError;

typedef struct t_gsfHVNavigationError
{
    struct timespec nav_error_time;
    int             record_id;          /* Containing nav with these errors */
    double          horizontal_error;    /* RMS error in meters */
    double          vertical_error;     /* RMS error in meters */
    double          SEP_uncertainty;    /* RMS error in meters */
    char            spare[2];           /* Two bytes reserved for future use */
    char            *position_type;     /* 4 character string code specifying type of
                                         positioning system */
}
gsfHVNavigationError;

```

4.1.10 Swath Bathymetry Summary Record

```

typedef struct t_gsfSwathBathySummary
{
    struct timespec start_time;
    struct timespec end_time;
    double          min_latitude;
    double          min_longitude;
    double          max_latitude;
    double          max_longitude;
    double          min_depth;
}

```

```

        double          max_depth;
    }
    gsfsWathBathySummary;

```

4.1.11 Attitude Record

```

typedef struct t_gsfsAttitude
{
    short          num_measurements;    /* number of attitude measurements in this
record */
    struct timespec *attitude_time;    /* seconds and nanoseconds */
    double         *pitch;             /* in degrees */
    double         *roll;              /* in degrees */
    double         *heave;            /* in meters */
    double         *heading;           /* in degrees */
}
gsfsAttitude;

```

4.2 Supporting Data Structures and Definitions

4.2.1 Record Identifier

```

typedef struct t_gsfsDataID
{
    int          checksumFlag;    /* boolean */
    int          reserved;        /* up to 9 bits */
    int          recordID;        /* bits 00-11 => data type number */
                                /* bits 12-22 => registry number */
    int          record_number;   /* specifies the nth occurrence of */
                                /* record type specified by recordID */
                                /* relevant only for direct access */
                                /* the record_number counts from 1 */
}
gsfsDataID;

```

4.2.2 Time Structure

```

struct timespec
{
    time_t      tv_sec;
    long        tv_nsec;
};

```

4.2.3 Null values used to represent missing data

```

/* Define null values to be used for missing data */
#define GSF_NULL_LATITUDE          91.0
#define GSF_NULL_LONGITUDE        181.0
#define GSF_NULL_HEADING          361.0
#define GSF_NULL_COURSE           361.0
#define GSF_NULL_SPEED            99.0

```

```

#define GSF_NULL_PITCH                99.0
#define GSF_NULL_ROLL                 99.0
#define GSF_NULL_HEAVE                99.0
#define GSF_NULL_DRAFT                0.0
#define GSF_NULL_DEPTH_CORRECTOR      99.99
#define GSF_NULL_TIDE_CORRECTOR        99.99
#define GSF_NULL_SOUND_SPEED_CORRECTION 99.99
#define GSF_NULL_HORIZONTAL_ERROR      -1.00
#define GSF_NULL_VERTICAL_ERROR        -1.00
#define GSF_NULL_HEIGHT                9999.99
#define GSF_NULL_SEP                  9999.99
#define GSF_NULL_SEP_UNCERTAINTY       0.0

/* Define null values for the swath bathymetry ping array types. Note that
 * these zero values do not necessarily indicate a non-valid value. The
 * beam flags array should be used to determine data validity.
 */
#define GSF_NULL_DEPTH                0.0
#define GSF_NULL_ACROSS_TRACK         0.0
#define GSF_NULL_ALONG_TRACK          0.0
#define GSF_NULL_TRAVEL_TIME          0.0
#define GSF_NULL_BEAM_ANGLE           0.0
#define GSF_NULL_MC_AMPLITUDE         0.0
#define GSF_NULL_MR_AMPLITUDE         0.0
#define GSF_NULL_ECHO_WIDTH            0.0
#define GSF_NULL_QUALITY_FACTOR        0.0
#define GSF_NULL_RECEIVE_HEAVE         0.0
#define GSF_NULL_DEPTH_ERROR           0.0
#define GSF_NULL_ACROSS_TRACK_ERROR    0.0
#define GSF_NULL_ALONG_TRACK_ERROR     0.0
#define GSF_NULL_NAV_POS_ERROR        0.0

```

4.2.4 Positioning System Type Codes

```

/* Define a set of macros that may be used to set the position type field */

#define GSF_POS_TYPE_UNKN "UNKN" /* Unknown positioning system type */
#define GSF_POS_TYPE_GPSU "GPSU" /* GPS Position, unknown positioning service */
#define GSF_POS_TYPE_PPSD "PPSD" /* Precise positioning service - differential */
#define GSF_POS_TYPE_PPSK "PPSK" /* Precise positioning service - kinematic */
#define GSF_POS_TYPE_PPSS "PPSS" /* Precise positioning service - standalone */
#define GSF_POS_TYPE_PPSG "PPSG" /* Precise positioning service - gypsy */
#define GSF_POS_TYPE_SPSD "SPSD" /* Standard positioning service - differential */
#define GSF_POS_TYPE_SPSK "SPSK" /* Standard positioning service - kinematic */
#define GSF_POS_TYPE_SPSS "SPSS" /* Standard positioning service - standalone */
#define GSF_POS_TYPE_SPSG "SPSG" /* Standard positioning service - gypsy */
#define GSF_POS_TYPE_GPPP "GPPP" /* Post Processing - Precise Point Positioning */
#define GPS_POS_TYPE_GPPK "GPPK" /* Post Processing - Post Processed Kinematic */

```