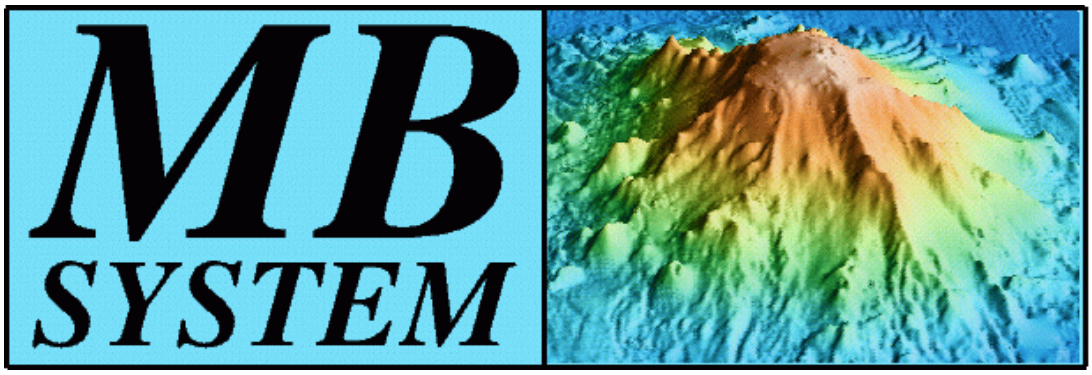


Developer's Guide to Coding an MB-System I/O Module

David W. Caress

Monterey Bay Aquarium Research Institute

Version 3

24 February 2014

Introduction

MB-System is an open source software package for the processing and display of swath mapping sonar data used to map the seafloor. The package consists of programs that manipulate, process, list, or display swath sonar bathymetry, amplitude, and sidescan data. The heart of the system is an input/output library called MBIO which allows programs to work transparently with any of a number of supported swath sonar data formats. This approach has allowed the creation of utilities that can be applied in a uniform manner to remote sensing data from a variety of sensors, mostly sonars. In order to allow MB-System to read and write

data in a great variety of native formats, the MBIO architecture allows for many separate input/output (I/O) modules. This document is intended to be a guide for writing a new MB-System I/O module and integrating that module with MB-System.

Diversity of Seafloor Mapping Data

Seafloor mapping data, whether measuring topography, imaging seafloor character, imaging subsurface structure, or a combination, derive from a great variety of sensors and are structured in many different forms. The relevant sensors are mostly sonars, and these can broadly be classed as single-beam echosounders, multibeam echosounders, sidescan sonars, interferometric sonars, and subbottom profilers. Since the first multibeam sonars became operational in the 1960's, hundreds of different sonar models have been used to map the seafloor, and the varied formats used to record digital data have been nearly as numerous.

The data from the different sonar classes (e.g. multibeam vs sidescan) can have very different structure (e.g. arrays of soundings from formed beams vs time series of backscatter from port and starboard staves), and even among similar sonars the key operational parameters and the detailed data structure can vary. Some mapping data formats consist of a single data record type repeated through entire files, and others interleave the primary sensor records with navigation, attitude, sound speed, and other types of records derived from ancilliary sensors. There are multiple data formats that contain essentially the same information (e.g. different formats storing data from the same sonar). There are also data formats that store data from multiple sensors, generally accomplishing this by leaving out some original information. There is also great variability in the underlying file types of data formats. Files may be constructed from ASCII text, records composed of binary integer and float values, or built out of architecture-independent representations such as XDR and netCDF. Most formats use single data files, but some represent data using multiple parallel files.

Because there are so many different forms of mapping data, it is infeasible to design a single "generic" data structure that can represent all of the currently used

data, or a single "generic" data format that can be used for all data. This problem is compounded by our very limited ability to predict the details of new systems that will become available and the new forms of data they will produce.

Modular Structure of MBIO

In order to support as broad a range of mapping data as possible without losing information, and to allow for advances in the remote sensing technology we use, we have architected MB-System to read and write data in the existing formats and to store those data internally with all information preserved. The consequence is that the MB-System input and output capability consists of a modular library (MBIO) supporting dozens of different seafloor mapping data formats. Each unique format is associated with functions that read the data into an internal representation, or data system, and write from that representation. A second level of modularity includes the many different data systems that are supported; each data system is defined by a structure used to store the data and a set of functions that extract commonly used values from, or insert values into, that structure.

In the terminology of this document, each MBIO I/O module consists of a single data system and at least one data format (Figure 1). Each data system includes both a structure to store the data and functions that map commonly used values to and from that structure. The data format includes functions that read and write the data to and from the data structure of the associated data system.

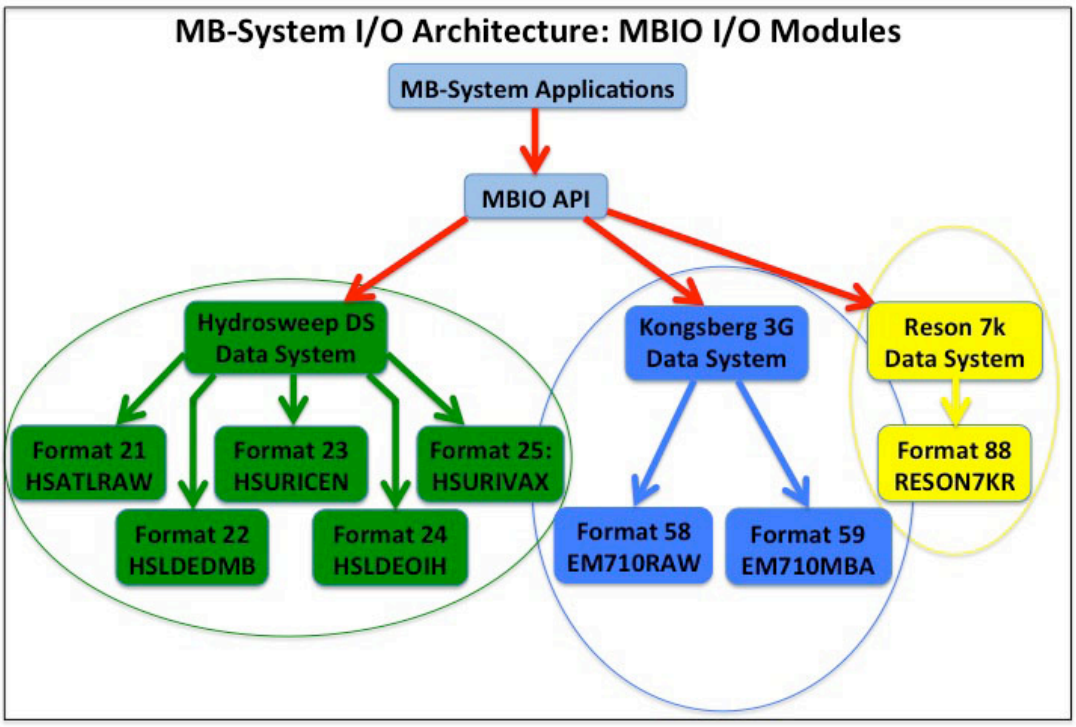


Figure 1. Schematic representation of MBIO structure with three I/O modules indicated by colored ellipses. Each I/O module consists of a single data system and its associated data formats. MB-System applications make calls to functions within the MBIO API. These are passed to the relevant data system, which can include one or more formats using the the same internal data storage structure. There is one set of data extraction and insertion functions for each data system (and I/O module), and separate read and write functions for each format within that I/O module.

Common API to read and write data in all supported formats

MB-System supports heterogenous data types by layering a common application programming interface (API) on top of the I/O modules that read and write sonar data in the existing data formats. The MBIO API consists of high level functions that allow applications to open data streams for reading or writing, read and write

data records sequentially, to straighforwardly extract and insert the commonly used values, and to expose the complete data representation to access. Although MB-System as a whole includes C, C++, and perl source code, the MBIO library is entirely written in C.

MBIO handles three types of swath mapping data: beam bathymetry, beam amplitude, and sidescan. Both amplitude and sidescan represent measures of backscatter strength. Beam amplitudes are backscatter values associated with the same preformed beams used to obtain bathymetry; MBIO assumes that a bathymetry value exists for each amplitude value and uses the bathymetry beam location for the amplitude. Sidescan is generally constructed with a higher spatial resolution than bathymetry, and carries its own location parameters. In the context of MB-System documentation, the discrete values of bathymetry and amplitude are referred to as "beams", and the discrete values of sidescan are referred to as "pixels". An additional difference between "beam" and "pixel" data involves data flagging. An array of "beamflags" is carried by MBIO functions which allows the bathymetry (and by extension the amplitude) data to be flagged as bad. The details of the beamflagging scheme are presented below.

MBIO opens and initializes sonar data files for reading and writing using the functions *mb_read_init()* and *mb_write_init()*, respectively. These functions return a pointer to a data structure including all relevent information about the opened file, the control parameters which determine how data is read or written, and the arrays used for processing the data as it is read or written. This pointer is then passed to the functions used for reading or writing. The structure (mb_io_struct{}) is defined in the file mbsystem/src/mbio/mb_io.h. There is no limit on the number of files which may be opened for reading or writing at any given time in a program. Both of the initialization functions call *mb_format_register()*, which in turn calls *mb_format_info()*.

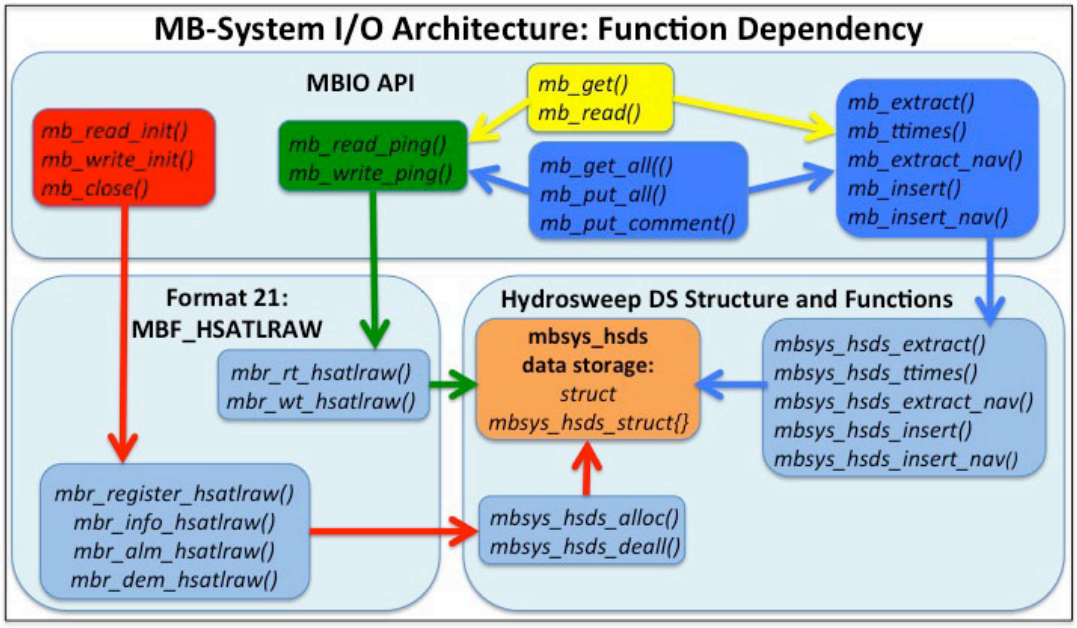
The *mb_read_init()* and *mb_write_init()* functions also return initial maximum numbers of bathymetry beams, amplitude beams, and sidescan pixels that can be used to allocate data storage arrays of the appropriate sizes. However, for some data formats there are no specified maximum numbers of beams and pixels, and so in general the required dimensions may increase as data are read.

Applications must pass appropriately dimensioned arrays into data extraction routines such as *mb_read()*, *mb_get()*, and *mb_get_all()*. In order to enable dynamic memory management of thse application arrays, the application must first register each array by passing the array pointer location to the function *mb_register_array()*.

Data files are closed using the function mb_close(). All internal and registered arrays are deallocated as part of closing the file.

When it comes to actually reading and writing swath mapping sonar data, MBIO has two levels of i/o functionality. The level 1 MBIO functions allow users to read sonar data independent of format, with the limitation that only a limited set of navigation information is passed. Thus, some of the information contained in certain data formats (e.g. the "heave" value in Hydrosweep DS data) is not passed by *mb_read()* or *mb_get()*. In general, the level 1 functions are useful for applications such as graphics which require only the navigation and the depth and/or backscatter values.

The level 2 functions (*mb_get_all()* and *mb_put_all()*) read and write the complete data structures, translate the data to internal data structures associated with each of the supported sonar systems, and pass pointers to these internal data structures. Additional functions allow a variety of information to be extracted from or inserted into the data structures (e.g. *mb_extract()* and *mb_insert()*). All information stored by a data system may be accessed directy using the structure definitions found in the data system header files. The great majority of processing programs use level 2 functions.




```
Figure 2. Function dependency within the MBIO API and between
the
API and a single data format in an I/O module. The initializat
ion
and closing functions (red) access format-specific functions t
hat
in turn access the data system and its data storage structure.
 The
level 1 (yellow) and level 2 (dark blue) reading and writing
functions (dark blue) access the format-specific reading and w
riting
functions to get data to and from the data system storage, and

then use the data extraction and insertion functions to access
 the
information in the storage structure.
```

An abbreviated description of the most important MBIO API functionality follows:

- **Level 1 reading**

  Level 1 functions are used for simple reading of swath data files. The primary

functions are:

- *mb_read()*
- *mb_get()*

The positions of individual beams and pixels are returned in longitude and latitude by *mb_read()* and in acrosstrack and alongtrack distances by *mb_get()*. Only a limited set of navigation information is returned. Comments are also returned. These functions can be used without any special include files or any knowledge of the actual data structures used by the data formats or MBIO.

- **Level 2 reading and writing**

  Level 2 functions provide complete reading and writing of data structures containing all of the available information. Data records may be read or written without extracting any of the information, or the swath data may be passed with the data structure. Several functions exist to extract information from or insert information into the data structures; otherwise, special include files are required to make sense of the sensor-specific data structures passed by level 2 i/o functions. The basic read and write functions that only pass pointers to internal data structures are:

  - *mb_read_ping()*
  - *mb_write_ping()*

  The read and write routines which both pass the data structure and extract or insert standard survey, navigation, or comment information are:

  - *mb_get_all()*
  - *mb_put_all()*
  - *mb_put_comment()*

  Once a pointer to the data structure is available following *mb_read_ping()* or *mb_get_all()*, other functions are available to extract or insert a variety of information. The extraction and insertion functions that are defined for all data systems (and therefore all I/O modules) are:

- *mb_extract()*
- *mb_insert()*
- *mb_ttimes()*
- *mb_detects()*
- *mb_extract_nav()*
- *mb_insert_nav()*
- *mb_extract_altitude()*
- *mb_insert_altitude()*
- *mb_copyrecord()*

The additional information extraction and insertion functions that are defined for only some data systems are:

- *mb_dimensions()*
- *mb_pingnumber()*
- *mb_segynumber()*
- *mb_sonartype()*
- *mb_sidescantype()*
- *mb_preprocess()*
- *mb_extract_nnav()*
- *mb_insert_altitude()*
- *mb_extract_svp()*
- *mb_insert_svp()*
- *mb_pulses()*
- *mb_gains()*
- *mb_extract_rawss()*
- *mb_insert_rawss()*
- *mb_extract_segytraceheader()*
- *mb_extract_segy()*
- *mb_insert_segy()*
- *mb_ctd()*
- *mb_ancilliarysensor()*

–

- **Format id numbers and information**

MBIO supports swath data in a number of different formats, each specified by a unique id number. The function *mb_format()* determines if a format id is valid. A set of additional functions returns information about the specified format:

- *mb_format_system()*
- *mb_format_dimensions()*
- *mb_format_description()*
- *mb_format_flags()*
- *mb_format_source()*
- *mb_format_beamwidth()*)

_

- **Verbosity**

  Most MBIO functions have an integer *verbose* as the first parameter. This value controls the degree to which the function prints out information to the stdout stream (verbose < 2) or the stderr stream (verbose >= 2). MBIO functions pass the *verbose* value on to any MBIO functions they call. MB-System programs allow multiple calls of the -V command line argument to set the verbose value greater than one. In general, the behavior can be characterized as:

  - *verbose* = 0: quiet
  - *verbose* = 1: verbose
  - *verbose* = 2: print a lot (all values on function entry and exit)
  - *verbose* = 3: overwhelm with information, direct this to a file...
  - *verbose* = 4: really overwhelm with information, direct this to a file
  - *verbose* = 5: really really overwhelm with information, all values read and written, direct this to a file

  _

- **Status and error values**

  Most MBIO functions return an integer value interpreted as the value *status*

and also have an integer pointer **error* as the last parameter. The possible *status* and *error* values are defined in mb_status.h. The *status* can be either MB_SUCCESS (status = 1) or MB_FAILURE (status = 0), with obvious meaning. The *error* value will be MB_ERROR_NO_ERROR if *status* = MB_SUCCESS, but may take on dozens of different values if *status* = MB_FAILURE. The most common meanings include:

- **error* = MB_ERROR_NO_ERROR = 0: No error - read a valid survey record
- **error* < 0: Non-fatal errors, reading and writing can continue
    - **error* = MB_ERROR_COMMENT: Comment record instead of a survey record
    - **error* = MB_ERROR_NAV: Navigation record instead of a survey record
    - **error* = MB_ERROR_UNINTELLIGIBLE: Unintelligible data read, but can keep reading
- **error* > 0: Fatal errors, reading and writing should be terminated
    - **error* = MB_ERROR_EOF: End of file
    - **error* = MB_ERROR_OPEN_FAIL: Unable to open file

_

- **Organizing data files with datalists**

  Most MB-System programs can process multiple data files specified in "datalist" files. Each line of a datalist file contains a file path and the corresponding MBIO format id. Datalist files can be recursive and can contain comments. The functions used to extract input swath data file paths from datalist files includes:

  - *mb_datalist_open()*
  - *mb_datalist_read()*
  - *mb_datalist_close()*

–

- **MB-System memory management tools**

  MBIO includes functions for allocating, reallocation, and deallocating memory that keep track of the allocated objects. This allows MBIO to deallocate all such objects when an input or output file is closed using *mb_close()*.

  - *mb_mallocd()*
  - *mb_reallocd()*
  - *mb_freed()*

–

- **Dynamic memory management for data arrays**

  The *mb_read_init()* and *mb_write_init()* functions return initial maximum numbers of bathymetry beams, amplitude beams, and sidescan pixels that can be used to allocate data storage arrays of the appropriate sizes. However, for some data formats there are no specified maximum numbers of beams and pixels, and so in general the required dimensions may increase as data are read. Applications must pass appropriately dimensioned arrays into data extraction routines such as *mb_read()*, *mb_get()*, and *mb_get_all()*. In order to enable dynamic memory management of thse application arrays, the application must first register each array by passing the array pointer location to the function:

  - *mb_register_array()*.

  Once arrays are registered, then whenever a data record is encountered that contains more beams or pixels than previously specified, those arrays are reallocated to accomodate the new amount of data. All registered arrays are deallocated when the associated data file is closed using the function mb_close().

- **On-the-Fly Merging of Asynchronous Navigation and Attitude**

Many data formats combine survey data records produced by sonars with navigation and attitude data records from other sensors, all sampled at different times. Sometimes the sonar interpolates the navigation and attitude values at the sonar ping time and includes these critical values in the survey data records. In cases where the format's survey records either do not store position and attitude at all (e.g. Kongsberg multibeam), or do not initially set the values (e.g. Reson 7k multibeam), the MB-System I/O module reading the data must interpolate (or extrapolate) the navigation and attitude at ping time from the separately read asynchronous navigation and attitude records.

This "on-the-fly" navigation and attitude merging is accomplished by having the *mb_rt_*()* function for the data format store values and timestamps as asynchronous data records are read, and then interpolate those values onto the ping timestamps as survey data records are read. This scheme uses the MBIO functions *mb_navint_add()*, *mb_navint_interp()*, *mb_attint_add()*, *mb_attint_interp()*, *mb_hedint_add()*, *mb_hedint_interp()*, *mb_depint_add()*, *mb_depint_interp()*, *mb_altint_add()*, and *mb_altint_interp()*.

- **Optimized File Reading and Writing**

  Some data formats have very large data records (>100 KB) that can be slow to read and write from a file system accessed across a network. In some cases, I/O performance can be enhanced by enlarging the data block buffer used by system *fread()* and *fwrite()* calls. MBIO supports modifying the i/o block buffer size for I/O modules that use *mb_fileio_open()*, *mb_fileio_read()*, *mb_fileio_write()*, and *mb_fileio_close()* in place of *fopen()*, *fread()*, *fwrite()* and *fclose()* . The modified block size is specified using the program mbdefaults.

The complete list of MBIO functions available to applications is included in the appendix, along with full prototypes and explanations. In addition to those introduced above, there are other MBIO functions dealing with default values for important parameters, error messages, memory management, and time conversions.

Organization of MB-System Source Code

The MB-System source code can be obtained either as a distribution "tarball" with a name something like mbsystem-5.4.2163.tar.gz, or by downloading directly from the source repository at http://svn.ilab.ldeo.columbia.edu/listing.php?repname=MB-System. After unpacking, the MB-System source will be structured as shown in Figure 3.

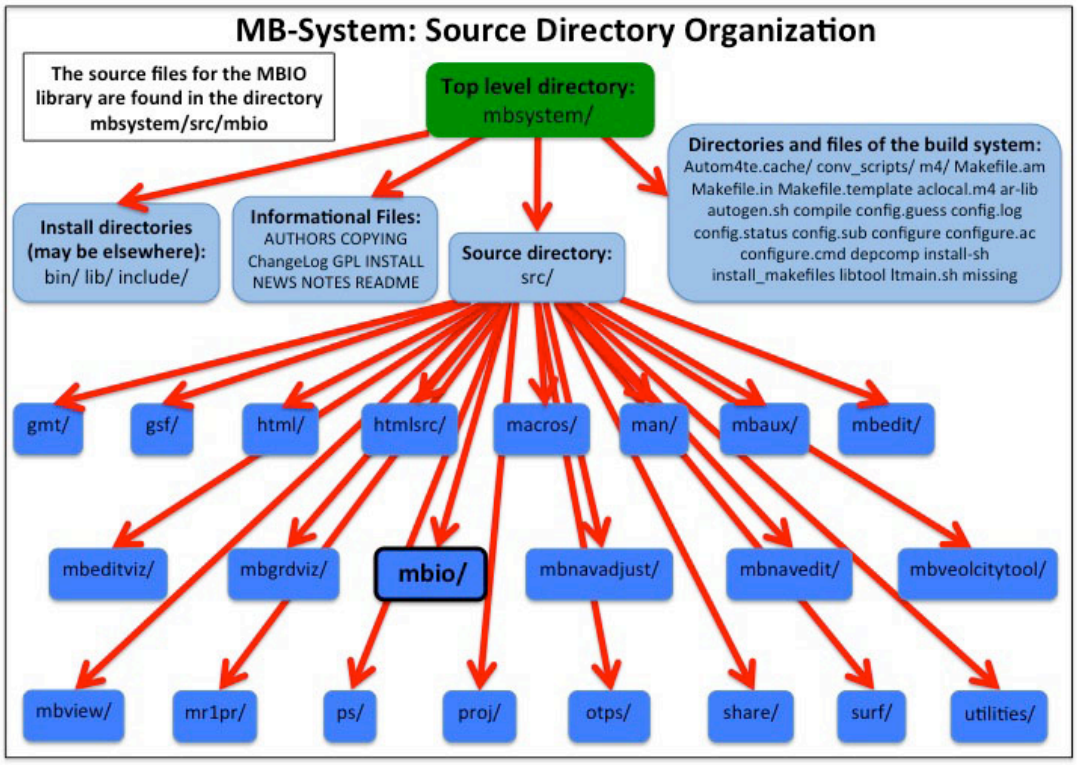


Figure 3. Schematic representation of the MB-System source code directory tree. All of the source files for libraries, programs, documentation, and supporting data are located under mbsystem/src. The source files for the MBIO library are entirely located within the mbsystem/src/mbio directory.

All of the source files associated with the MBIO library are found in mbsystem/src/mbio. In order to implement support for a new I/O module in MB-System, one will have to write some new source files that must reside in mbsystem/src/mbio, and modify a few of the existing files. No changes to files

outside of mbsystem/src/mbio are required to support either a new format tied to an existing data system or an entire new I/O module.

The directories other than mbio under mbsystem/src contain all of the other source code, supporting data, and documentation comprising the MB-System package. The source files for non-graphical MB-System programs such as mbinfo and mbprocess are found mbsystem/src/utilities. The source files for the GMT-compatible programs mbcontour and mbswath are located in mbsystem/src/gmt, and all of the graphical utilities such as mbedit and mbgrdviz have their own eponymous source directories (e.g. mbsystem/src/mbedit). A number of specialized support libraries are sourced in mbsystem/src/mbaux. Three data formats, GSFGENMB (MBIO id 121), SURFSAME (MBIO id 181), and MR1PRHIG (MBIO id 61) are supported by externally supplied i/o libraries that are sourced in the dedicated directories mbsystem/src/gsf, mbsystem/src/surf, and mbsystem/src/mr1pr, respectively. The I/O modules for these formats include the usual source files in mbsystem/src/mbio described below, but in these cases the low level calls to read and write data are made to the specialized libraries. MB-System documentation source files are located in the mbsystem/src/man, mbsystem/src/htmlsrc, mbsystem/src/html, and mbsystem/src/ps directories. Supporting data, specifically a cartographic projection list and a global water sound speed database, are located in /mbsystem/src/share.

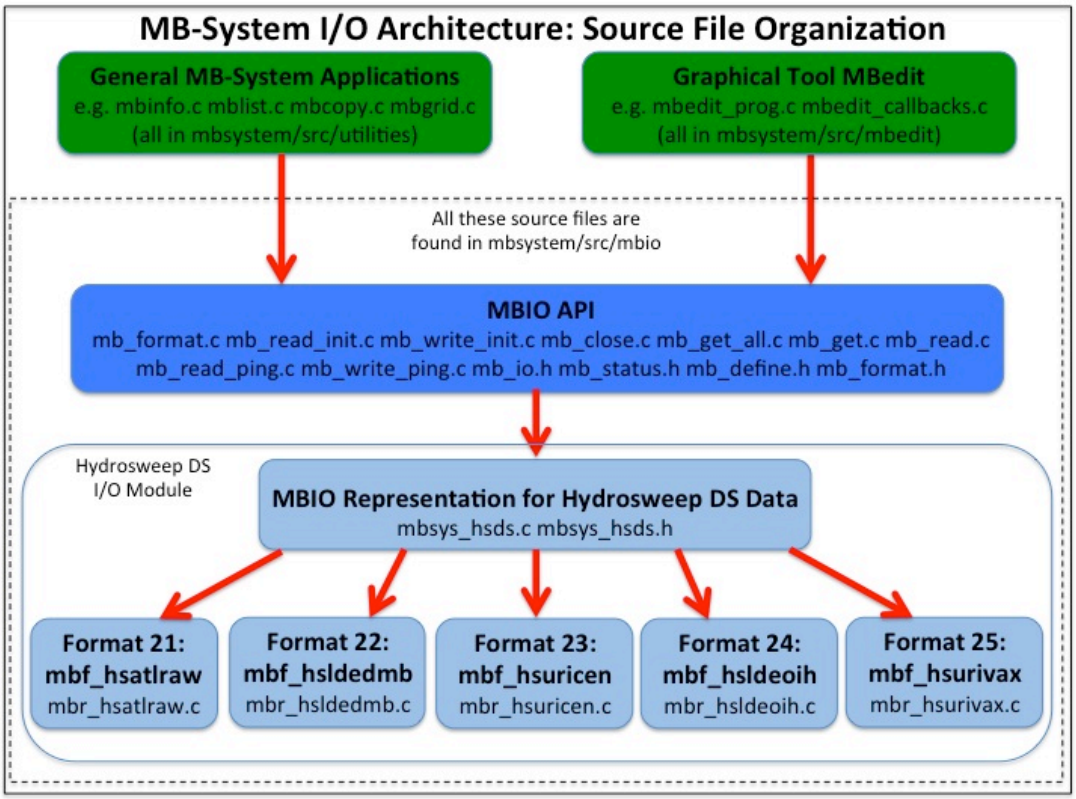


Figure 4. Organization of MBIO source files by name for one of the I/O modules. The source files for MB-System programs (e.g. mbinfo.c) that make calls to the MBIO library are found in other directorys under mbsystem/src/. All of the source files for the MBIO library are found in the mbsystem/src/mbio/ directory. The API functions are defined in source files with names of the form mb_*.c, such as mb_format.c. The data system aspects of the Hydrosweep DS multibeam I/O module are found in the files mbsys_hsds.c and mbsys_hsds.h, where hsds is a reasonably informative shorthand name for the data system. The reading and writing functions for format HSLDEOIH, the Lamont-Doherty in house binary format for Hydrosweep DS data (MBIO format 24), are found in the file mbr_hsldeoih.c. The files mbsys_hsds.c, mbsys_hsds.h, mbr_hsldeoih.c and the other mbr_*.c files associated with the HSDS data system comprise the I/O module for Hydrosweep DS data.

Figure 4 provides a sense (albeit incomplete) of the C source file names and structure associated with the MBIO library. Within the mbsystem/src/mbio directory, there are *.c C files with names of the form mb_*.c that include the MBIO library functions accessible to all MB-System programs. Examples include mb_read_init.c, which includes the code for function mb_read_init(), mb_format.c, which includes several functions supporting the interfacing of data formats with the API, and mb_access.c, which includes many of the functions used to extract and insert information from and to data structures. There are also *.h header files with names of the form mb_*.h that include the function prototypes, structure definitions, and macro definitions needed for functions and programs to make use of the MBIO library. All MB-System program source files must reference at least these header files:

- mb_status.h
- mb_define.h
- mb_format.h

and many also reference one or both of:

- mb_io.h
- mb_process.h

The data system aspects of an I/O module are found in files named mbsys_XXXX.c and mbsys_XXXX.h, where XXXX is a reasonably informative shorthand name for the data system (e.g. hsds for Hydrosweep DS, reson7k for Reson 7000 series multibeams). The XXXX string can be any length of characters. The data format reading and writing functions are found in files named mbr_YYYYYYYY.c, where YYYYYYYY is an eight character shorthand name for the format (e.g. hsldeoih for the Lamont-Doherty in house binary format for Hydrosweep DS data, or reson7kr for the Reson 7k vendor format for Reson 7000 series multibeam data). For a give data system XXXX and all associated data formats YYYYYYYY, the mbsys_XXXX.c, mbsys_XXXX.h, and all associated mbr_YYYYYYYY.c files comprise a single I/O module.

Example of a New MB-System I/O Module

The remainder of this document will be concerned with a specific example of writing an MB-System I/O module. The data to be supported derive from WASSP multibeam sonars in the data format output by the WMB-3230, WMB-3250, and WMB-5230 models. These sonars are designed and sold by Electronic Navigation Ltd (ENL)under the brand WASSP Ltd. This format is only relevant for data files created by logging the realtime records without modification; some commercial datalogger systems will record in different proprietary formats. The specification for the relevant native format can be found at: http://www.mbari.org/data/mbsystem/formatdoc/WASSP_Generic_ICD_2.2.pdf Using WASSP data in this context does not imply any endorsement of the sonar, data produced by the sonar, or the sonar vendor. The WASSP data have been chosen for this example because WASSP data have not been previously supported in MB-System, and the data format is neither trivially simple nor excessively complicated.

We have two WASSP data samples. One comes from ENL and is logged using software that they use for testing but do not sell. The other comes from Jonathan Beaudoin of the University of New Hampshire Center for Coastal and Ocean Mapping (UNH/CCOM), and was logged using his own software.

The ENL sample is a file:

```
-rw-rw-r--  675641961  GNLogger.nwsf
```

The UNH/CCOM sample is a file:

```
-rw-rw-r--   42891976  20131107_165148.000
```

We will primarily depend on the UNH/CCOM data sample for the development and testing of this I/O module.

I/O Module Source Files

The source files for this document's WASSP multibeam example I/O module are in the mbsystem/src/mbio/ directory (and in Appendix 2 of this document) and are

named:

- mbsys_wassp.h
- mbsys_wassp.c
- mbr_wasspenl.c

The discussion below includes sections of these source files, but readers will doubtless need to refer to these files in their entirety.

I/O Module Template Files

Templates for mbsys_*.c, mbsys_*.h, and mbr_*.c files are also available in the mbsystem/src/mbio directory (and in Appendix 3 of this document). These are:

- mbsys_templatesystem.c
- mbsys_templatesystem.h
- mbr_tempform.c

These files can be copied, renamed, and used as the basis for coding new I/O modules for MB-System.

Overview of Coding an I/O Module

This section provides a short list overview of the steps to creating a new I/O module. These steps are discussed in detail in the sections below.

In order to write a new MB-System I/O module and integrate it with MBIO, do the following:

1. Select the data system name for the new I/O module, along with a name and id number for each data format.
    - In this example the data system will be "WASSP"." The single data format will be named "wasspenl" and have a format id of 241.

2. Copy the template files to generate appropriately named source files for the I/O module.
    - In this example the I/O module files will be: mbsys_wassp.h,

mbsys_wassp.c, and mbr_wasspenl.c.

3. Write the data structure (or structures) in file mbsys_*.h that are required to represent all of the data in the relevant formats.
4. Fill in the functions for reading and writing the data in the mbr_*.c files for all of the relevant formats.
5. Fill in the functions in file mbsys_*.c that are required to extract information from and insert information into the data structure(s) defined in mbsys_*.h.
6. Integrate the new I/O module by modifying a few other key source files in mbsystem/mbio.
    - Add to the lists of data systems in mb_format.h.
        - Add: #define MB_SYS_WASSP 36
    - Add to the lists of data formats in mb_format.h.
        - Add: #define MBF_WASSPENL 241
        - Increment #define MB_FORMATS 75
    - Add prototype data format registration functions in mb_format.h.
        - Add prototypes for functions mbr_register_wasspenl() and mbr_info_wasspenl()
    - Add references to the new formats in the following functions found in mb_format.c:
        - mb_format_register()
        - mb_format_info()
        - mb_get_format()
7. Update the MB-System build system to include the new source files.
8. Test the use of the new I/O module

Step 1: Select the Data System Name and Data Format Names and ID's

The data system and format names of an I/O module should be both unique (within MB-System) and moderately meaningful.

The data system name can be of any length, though a reasonable upper bound might be 25 characters. Examples include "hsds" for Hydrosweep DS multibeam data, "SB2100" for SeaBeam 2112/2120/2136 multibeam data, and RESON7K for Reson 7000 series multibeam data.

Data format names must be eight characters long (this insures proper formatting of some tables). Every format also has a unique MBIO format id number, and when there are multiple formats in an I/O module the format id numbers should be sequential. Figures 1 and 4 show the MBIO names and format id's for five Hydrosweep DS multibeam formats associated with the HSDS data system: HSATLRAW (21), HSLDEDMB (22), HSURICEN (23), HSLDEOIH (24), HSURIVAX (25). In order to allow for new formats to be added to existing I/O modules, the starting id number for a new I/O module is always ten greater than for the previous one, and is a multiple of ten plus one. The current highest numbered format is "MBF_3DDEPTHP" with an id number of 231. Consequently, the next I/O module will have a first format id number of 241.

Since the data to be supported in this example derive from the WASSP multibeam sonars produced by ENL, we chose to name the new data system "WASSP" and the first format "WASSPENL". As just noted, "WASSPENL" will have a format id of 241.

Step 2: Prepare the Source Files from the Templates

Template versions of the basic MBIO I/O module files are included in the MB-System source distribution. These files are in the mbsystem/src/mbio directory, and have the names:

- mbsys_templatesystem.h
- mbsys_templatesystem.c
- mbr_tempform.c

To make use of these files, we replace the existing data system name (templatesystem) and format name (tempform) by the new names, and also rename the files accordingly. Since the new data system is named "WASSP"" and the new format is named "WASSPENL", we make the following changes:

- Copy mbsys_templatesystem.h to new file mbsys_wassp.h
- Copy mbsys_templatesystem.c to new file mbsys_wassp.c
- Copy mbr_tempform.c to new file mbr_wasspenl.c
- Within all three files, globally change "templatesystem" to "wassp"
- Within all three files, globally change "TEMPLATESYSTEM" to "WASSP"
- Within all three files, globally change "tempform" to "wasspenl"
- Within all three files, globally change "TEMPFORM" to "WASSPENL"

Once the I/O module files have been recast to the new names, we can begin to write the actual code to read, write, store, and access the new data.

Step 3: Define Data Structures Required to Store the New Data

The data structure used to store the WASSP multibeam data is defined in the file mbsys_wassp.h. Since the data system is named "WASSP", the primary data structure is named

```
struct mbsys_wassp_struct{};
```

For some I/O modules, the primary data structure contains all of the relevant variables without any sub-structures. In other cases, there is a separate data structure defined for each data record type, and the primary data structure is essentially a holder for the various sub-structures. The approach used for any particular I/O module is chosen by the developer, presumably to improve the readability and maintenability of the code.

In the case of the WASSP multibeam data, we find from the ICD document that these data files can contain at least six data record types, four of which may be produced by each survey ping, and two of which are asynchronous with the survey data. The records associated with survey data are named "GENBATHY", "CORBATHY", "RAWSONAR", and "WCD_NAVI", and the asynchronous records are named "NVUPDATE" and "GEN_SENS".

As one begins to code a data format, it is important to examine the sample data to

verify one's understanding of the format and to identify differences between the specification document and the actual data. In general, most format documentation contains some ambiguities and errors; a working I/O module must be coded to the actual form of the data, not the documented form of the data. Since in this case we have data samples from two sources, we need to closely inspect both.

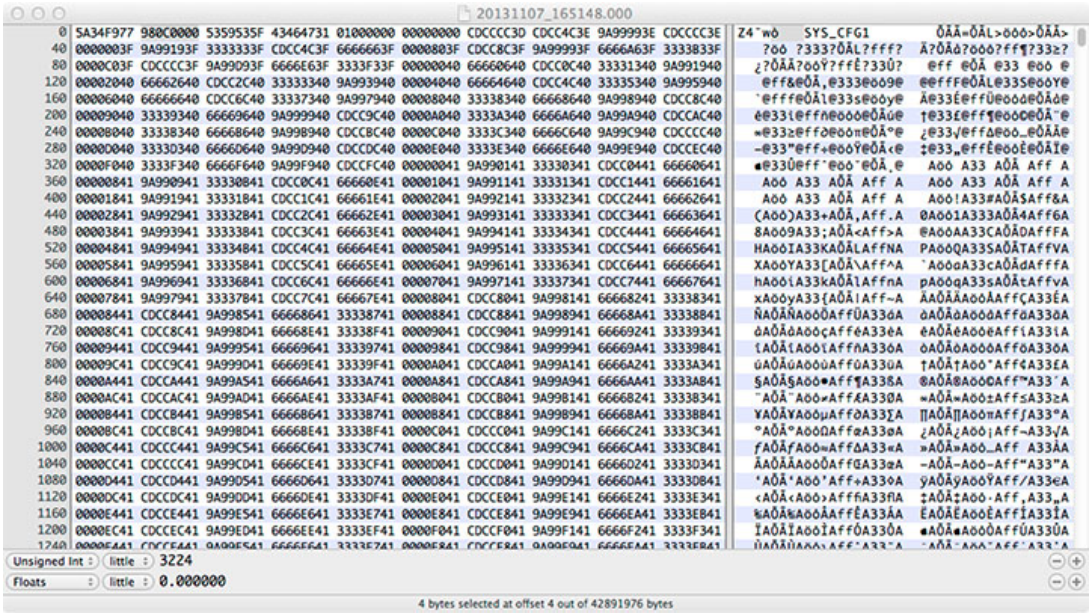


Figure 5. View of the program Hex Fiend showing the start of the UNH/CCOM data sample file 20131107_165148.000. The first 16 bytes form the header of the first data record. The 4-byte sync value is 0x5A34F977. The next four bytes are an unsigned int value of 3224, corresponding to the data record size in bytes. The following 8 bytes is the data record name as UTF-8 characters, here showing "SYS_CFG1".

This record type is not listed in the ICD document.

A good hexadecimal editor allows one to interactively parse a complex binary data file. Figure 5 shows the beginning of the UNH/CCOM data sample as displayed by the program Hex Fiend. This program views the data in both hexadecimal and text forms, and also shows the integer and floating point values of selected byte groups. In this case, we see that the first four bytes have the form 0x5A34F977,

rather than the 0x77F9345A listed in the ICD. This apparent reversal of the byte order is a result of the "little-endian" structure of these data; the sync value listed is that of an unsigned four byte integer represented in little-endian form. The second four-byte unsigned integer gives the record size of 3224 bytes. The next eight bytes are the data record identifier as characters, here showing as "SYS_CFG1". This is not a record type listed in the ICD!

Thus, we know that our I/O module code will need to handle at least one undocumented record, and probably needs to be able to handle arbitrary unknown records gracefully. In MB-System, handling an unknown data record gracefully, means reading it all, storing it all, and writing it back out to the output file unchanged, all without knowing anything about the contents.

For binary data formats, it is critical to understand the byte-order (little-endian vs big-endian) used in the data files. In the case of the WASSP data, the ICD indicates that integer and float values of all sizes are represented using little-endian byte order. Some data formats allow both types of byte ordering, typically determined by the type of computer used for sonar control and/or data logging. In such cases the I/O module code must be capable of discerning the byte order of the data it reads.

A thorough investigation of both data samples and the ICD yields a number of comments and issues with regard to reading and writing these data. These sorts of comments should be embedded in the top of the I/O module header file, in this case mbsys_wassp.h. For these data, the initial comments are:

```
/*
 * Notes on the mbsys_wassp data structure and associated form
at:
 *   1. This MBIO format supports the generic interface messag
e output
 *      by the ENL WASSP multibeam sonars.
 *   2. Reference: WASSP Generic ICD 2.2, 15 October 2013.
 *   3. The WASSP data stream consists of several different da
ta
 *      records that can vary among models and installations.
 *   4. The WASSP multibeam models as of January 2014 incude:
```

```
 *        WMB-3230, 160 kHz, 112 beams, 120 degree swath
 *        WMB-5230, 80 kHz, 112 beams, 120 degree swath
 *        WMB-3250, 160 kHz, 224 beams, 120 degree swath
 *   5. The alongtrack beamwidths are 4 degrees, the acrosstra
ck
 *      beamwidths are estimated to be 2 degrees since the tra
nsducer
 *      arrays are about two times wider than long.
 *   6. Each data record begins with a 16 byte sequence:
 *        unsigned int sync;        \\ 0x77F9345A
 *        unsigned int size;        \\ Size in bytes of this r
ecord from start
 *                                  \\     of sync pattern to
end of checksum
 *        char         header[8];   \\ Data record name
 *   7. All data are in little-endian form.
 *   8. The data record names include:
 *        GENBATHY - Uncorrected Bathymetry
 *        GEN_SENS - External Sensor Data
 *        RAWSONAR - Raw water column data (roll stabilized so
nar)
 *        CORBATHY - Corrected Bathymetry
 *        NVUPDATE - Nav Data Message
 *        WCD_NAVI - Water Column Information
 *        SYS_CFG1 - Unknown record at start of file
 *                     (3224 bytes including sync and checksum
)
 *   9. A sample file logged by the ENL datalogger has an 804
byte UTF-8
 *      header that looks like:
 *      ************************************************
************************
 *      ************************************* PM Version: 2
.4.1.288***************
 *      ************************************* GUI Version:
2.4.1.125***************
 *      ************************************* Mity Verion:
20060***************
 *      ************************************************
************************
```

```
 *      ************************************************************
************************
 *      ************************************** PM Version: 2
.4.1.288****************
 *      ************************************** GUI Version:
2.4.1.125****************
 *      ************************************** Mity Verion:
20060****************
 *      ************************************************************
************************
 *      The reading code will search for valid sync values and
 so should ignore similar
 *      headers when encountered.
 *      A sample file logged by Jonathon Beaudoin of the Unive
rsity of New Hampshire
 *      Center for Coastal and Ocean Mapping does not contain
this header, so it is a
 *      feature of the ENL datalogger and not part of the data
 stream output by the
 *      WASSP sonars.
 *  10. The CORBATHY records do not have a full time stamp, an
d thus do not stand alone
 *      as survey data. Both a GENBATHY and a CORBATHY record
are required to form a useful
 *      survey record.
 *  11. A survey record may or may not include a RAWSONAR reco
rd and a WCD_NAVI record.
 *  12. The UNH/CCOM data samples contain adjacent GENBATHY an
d CORBATHY records for each ping.
 *      The ENL-logged data have separate groups of GENBATHY r
ecords and CORBATHY records,
 *      with each CORBATHY record occuring 10 to 30 records af
ter the corresponding
 *      GENBATHY record. Consequently, parsing these data woul
d require buffering the GENBATHY
 *      records to match them with the right CORBATHY record.
The current I/O module is
 *      not implemented with GENBATHY buffering, and will not
work with the ENL data sample.
 *  13. The contents of the GEN_SENS records are not specified
```

```
 and are presently unknown.
 *  14. The NVUPDATE record does not include a time stamp and
thus does not usefully serve
 *      as asynchronous navigation and attitude.
 *  15. The maximum number of beams is known, and can be stati
cally dimensioned for the
 *      GENBATHY and CORBATHY records. However, the maximum nu
mbers of samples in the RAWSONAR
 *      and WCD_NAVI records are not defined, and so these str
uctures must allow for
 *      dynamic allocation of memory.
 *
 */
```

Once some knowledge of the ideosyncrasies of the actual data format are known, one can make decisions about how to best represent the data in memory. Since there are a number of different data record types and some similar values appearing in more than one record type (e.g. time, ping number), we choose to define separate sub-structures for each data record type. The primary data structure has the form:

```
/* Data system structure */
struct mbsys_wassp_struct
    {
    /* Type of most recently read data record */
    int     kind;           /* MB-System record ID */

    /* MB-System time stamp of most recently read record */
    double      time_d;
    int     time_i[7];

    /* GENBATHY record */
    struct mbsys_wassp_genbathy_struct genbathy;

    /* CORBATHY record */
    struct mbsys_wassp_corbathy_struct corbathy;

    /* RAWSONAR record */
    struct mbsys_wassp_rawsonar_struct rawsonar;

    /* GEN_SENS record */
    struct mbsys_wassp_gen_sens_struct gen_sens;

    /* NVUPDATE record */
    struct mbsys_wassp_nvupdate_struct nvupdate;

    /* WCD_NAVI record */
    struct mbsys_wassp_wcd_navi_struct wcd_navi;

    /* SYS_CFG1 record */
    struct mbsys_wassp_sys_cfg1_struct sys_cfg1;

    /* MCOMMENT Record */
    struct mbsys_wassp_mcomment_struct mcomment;

    /* unknown Record */
    struct mbsys_wassp_unknown1_struct mcomment;
    };
```

The structure begins with three critical parameters that must exist in every MBIO

primary data structure. The *kind* value indicates the type of data record most recently read into this structure. Over sixty data record types are defined in mb_status.h; the most common record types are:

- MB_DATA_DATA Survey data
- MB_DATA_NAV Navigation data
- MB_DATA_COMMENT Comment string

The *time_d* value is the time stamp of the most recent record in the first MB-System standard form, which is seconds since the beginning of the year 1970. This is also commonly known as "Unix" time or epoch time. The *time_i* value represents the same time stamp in the second MB-System standard form, which is as an array of seven integers where:

- time_[0] ==> year (e.g. 2014)
- time_[1] ==> month (e.g. 2)
- time_[2] ==> day (e.g. 16)
- time_[3] ==> hour (e.g. 12)
- time_[4] ==> minute (e.g. 18)
- time_[5] ==> second (e.g. 49)
- time_[6] ==> microsecond (e.g. 350627)

The MBIO functions *mb_get_time()* and *mb_get_date()* translate between the *time_d* and *time_i* representations.

The sub-structures associated with the individual data records are defined above the primary structure in mbsys_wassp.h. The two most important structures are those holding the values read from GENBATHY and CORBATHY data records:

```
/* Individual data record structures */
struct mbsys_wassp_genbathy_struct
        {
        /* GENBATHY Record */
        /* Uncorrected Bathymetry
         * All the bottom detection points will be supplied as
 range and angle values.
         * The length of the output message is variable, depen
```

```
dant on the number of beam data.
         * In addition to the Flags the sample number will be
set to zero when detection is invalid. */
        unsigned int    version;               /* 3 */
        double          msec;                  /* A millisecon
d time stamp of rising edge
                                                          * of
Transmit pulse in UTC time (UTC time
                                                          * is
calculated from the timestamp of the
                                                          * ZDA
 sentence and or a PPS signal when
                                                          * ava
ilable) No local time zone correction
                                                          * is
applied. */
        mb_u_char       day;                   /* UTC time fro
m NMEA ZDA */
        mb_u_char       month;                 /* UTC time fro
m NMEA ZDA */
        unsigned short  year;                  /* UTC time fro
m NMEA ZDA */
        unsigned int    ping_number;           /* Sequential n
umber. */
        unsigned int    sonar_model;
        unsigned long   transducer_serial_number;
        unsigned int    number_beams;
        unsigned int    modeflags;
        float           sampling_frequency;    /* Hz */
        float           acoustic_frequency;    /* Hz */
        float           tx_power;              /* Voltage (vol
ts) rms applied to
                                                          * tra
nsmitter in dB.*/
        float           pulse_width;           /* Pulse width
in milliseconds */
        float           absorption_loss;       /* Configurable
 value applied by WASSP. */
        float           spreading_loss;        /* 0, 30 or 40
as selected by WASSP GUI,
```

```
                                                                * uni
ts dB (as function of target range
                                                                * in
metres) */
        unsigned int    sample_type;           /* Set to 0 if
un-calibrated.
                                                                * Set
 to 1 if calibrated. */
        float           sound_velocity;        /* Sound veloci
ty at the sonar head in m/s
                                                                * (th
at was used in beam forming) */
        float           detection_point[MBSYS_WASSP_MAX_BEAMS]
;
                                                               /* Non
-corrected fractional sample number
                                                                * wit
h the reference to the receiver's
                                                                * aco
ustic centre with the zero sample at
                                                                * the
 transmit time. */
        float           rx_angle[MBSYS_WASSP_MAX_BEAMS];
                                                               /* Bea
m steering angle with reference to
                                                                * rec
eiver's acoustic centre in the sonar
                                                                * ref
erence frame, at the detection point;
                                                                * in
degrees. */
        unsigned int    flags[MBSYS_WASSP_MAX_BEAMS];
                                                               /* Bit
 fields:
                                                                   Bi
t 0: Detection Success
                                                                   Bi
t 1: Detection Valid
                                                                   Bi
t 2: WMT Detection
```

```
                                                                    Bi
t 3: SAC Detection
                                                                    Bi
ts 4-31: Reserved for future use */
        float           backscatter[MBSYS_WASSP_MAX_BEAMS];
                                                            /* Max
 target strength (dB) at seafloor on
                                                             * thi
s beam. */
        unsigned int    checksum;               /* checksum for
mat TBD */
        };

struct mbsys_wassp_corbathy_struct
        {
        /* CORBATHY Record */
        /* Corrected Bathymetry
         * Use this data for use as fully corrected bathymetry
 data. The ships sensors
         * integrated into the WASSP correct this information
for leaver arm, pitch, roll, yaw,
         * heave, tide etc. Each total message contains the de
tections data for a single ping.
         * NOTES:
         *      1) Sign of Latitude (N = +ve)
         *      2) Sign of Longitude (E = +ve)
         *      3) All points are sent for every beam even if
they contain no detection data.
         *         So you can check if it is valid by checking
 the Y value, if this is 0 then the
         *         detection is not valid and should not be us
ed.
         *      4) The X,Y,Z positions are based on the fully
corrected output using leaver arm,
         *         sensor data and sound speed information ava
ilable. This means if the X, Y, Z
         *         offsets in the WASSP application are correc
t, there is no need to account for
         *         the distance between GPS antenna and transd
ucer or any pitch/roll/heave inclination.
```

```
         *      5) Sign of Longitude is normal (East = positiv
e)
         *      6) Depths are tide corrected unless tides are
disabled in the WASSP system.
         */
        unsigned int    version;               /* 3 */
        double          msec;                  /* A millisecon
d time stamp of rising edge
                                                * of
Transmit pulse in UTC time (UTC time
                                                * is
calculated from the timestamp of the
                                                * ZDA
 sentence and or a PPS signal when
                                                * ava
ilable) No local time zone correction
                                                * is
applied. */
        unsigned int    num_beams;             /* Fixed by sof
tware. Invalid points have depth set to 0.0. */
        unsigned int    ping_number;           /* Ping sequenc
e number */
        double          latitude;              /* Latitude at
transducer in degrees*/
        double          longitude;             /* Longitude at
 transducer in degrees */
        float           bearing;               /* Bearing/Head
ing of vessel on transmit in degrees */
        float           roll;                  /* Roll of vess
el on transmit in radians */
        float           pitch;                 /* Pitch of ves
sel on transmit in radians */
        float           heave;                 /* Heave of ves
sel on transmit at transducer in meters */
        unsigned int    sample_type;           /* Set to 0 if
un-calibrated. Set to 1 if calibrated. */
        unsigned int    spare[6];
        unsigned int    beam_index[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Bea
m index number */
```

```
        float           x[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Dis
tance to detection point in metres laterally along
                                                         * wes
t/east axis. East of vessel is positive. */
        float           y[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Dis
tance to detection point in metres laterally along
                                                         * nor
th/south axis. South of vessel is positive. */
        float           z[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Dep
th in meters for the detection point. -ve = down.
                                                         * 0 =
 not valid */
        float           beam_angle[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Ang
le of the beam this detection appears on in radians
                                                         * pos
itive for starboard side of vessel. */
        float           backscatter[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Max
 target strength (dB) at seafloor on this beam. */
        mb_u_char       quality[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Det
ection information - (0=none, 1=WMT, 2=SAC) */
        mb_u_char       fish[MBSYS_WASSP_MAX_BEAMS];
                                                        /* Fis
h intensity value for all fish targets vertically above
                                                         * det
ection point. */
        mb_u_char       roughness[MBSYS_WASSP_MAX_BEAMS]; /* U
nused */
        mb_u_char       empty[MBSYS_WASSP_MAX_BEAMS];   /* Unu
sed */
        unsigned int    pad[MBSYS_WASSP_MAX_BEAMS];     /* Unu
sed */
        unsigned int    checksum;               /* checksum fo
rmat TBD */
       };
```

As noted above, the RAWSONAR and WCD_NAVI records require dynamic storage, and so the corresponding structures contain pointers. The arrays attached to these pointers must be allocated large enough to hold the data read, whatever that turns out to be.

```
struct mbsys_wassp_rawsonar_struct
        {
        /* RAWSONAR Record */
        /* Raw water column data (roll stabilized sonar)
         * This packet is only roll stabilized if the WASSP sy
stem has valid roll information available.
         * The data contained in this packet is to be dB signa
l levels received by time and angle. Some
         * filtering of the data may be applied to remove side
 lobes and noise. This data is likely to be
         * a subset of the full sampling resolution of the sys
tem – less beams and less samples. The first
         * sample of raw data is the first sampling period sta
rting from the rising edge of the transmit
         * pulse and ending at the end of the sampling period
determined by the Sample Rate.
         */
        unsigned int    version;              /* 2 */
        double          msec;                 /* A millisecon
d time stamp of rising edge
                                                  * of
Transmit pulse in UTC time (UTC time
                                                  * is
calculated from the timestamp of the
                                                  * ZDA
 sentence and or a PPS signal when
                                                  * ava
ilable) No local time zone correction
                                                  * is
applied. */
        unsigned int    ping_number;          /* Sequential n
umber. */
        float           sample_rate;          /* Frequency (H
```

```
z) of raw data in this packet */
        unsigned int    n;                      /* Number of be
ams of raw data in this packet */
        unsigned int    m;                      /* Number of sa
mples (per beam) of raw data in this packet */
        float           tx_power;               /* Voltage (vol
ts) rms applied to
                                                 * tra
nsmitter in dB.*/
        float           pulse_width;            /* Pulse width
in milliseconds */
        unsigned int    sample_type;            /* Set to 0 if
un-calibrated.
                                                 * Set
 to 1 if calibrated. */
        unsigned short  spare[MBSYS_WASSP_MAX_BEAMS];
                                                /* Set
 to 0 until assigned a function */
        unsigned short  beam_index[MBSYS_WASSP_MAX_BEAMS];
                                                /* Equ
ivalent beam Index into uncorrected bathy
                                                 * (GE
NBATHY) record of each beam. */
        unsigned int    detection_point[MBSYS_WASSP_MAX_BEAMS]
;
                                                /* Ind
ex of sample which most closely matches
                                                 * sea
floor detection. 0 = not valid. */
        float           beam_angle[MBSYS_WASSP_MAX_BEAMS];
                                                /* Bea
m angle for this beam in degrees (negative
                                                 * por
t side of nadir) */
        size_t          rawdata_alloc;          /* Number of sh
orts allocated for rawdata array */
        short           *rawdata;               /* If Sample Ty
pe = 0 then Signal Levels at
                                                 * sam
ple/beam in dB*100 (divide by 100 to get
```

```
                                                                * act
ual signal level dB). The order is N x sample 1
                                                                * the
n N x sample 2... etc. If Sample Type = 1
                                                                * the
n calibrated db*100. */
        unsigned int    checksum;               /* checksum for
mat TBD */
       };

struct mbsys_wassp_wcd_navi_struct
        {
        /* WCD_NAVI Record */
        /* Water Column Information
         * This message is sent over the network after each de
tection message is sent, thus the water
         * column data is valid for the previous ping that has
 just been received. */
        unsigned int    version;                /* 3 */
        double          latitude;               /* Latitude fro
m GPS sensor in decimal degrees */
        double          longitude;              /* Longitude fr
om GPS sensor in decimal degrees */
        unsigned int    num_points;             /* Number of wa
ter column points to follow */
        float           bearing;                /* Bearing of v
essel for fish targets, degrees */
        double          msec;                   /* A millisecon
d time stamp of rising edge
                                                                * of
Transmit pulse in UTC time (UTC time
                                                                * is
calculated from the timestamp of the
                                                                * ZDA
 sentence and or a PPS signal when
                                                                * ava
ilable) No local time zone correction
                                                                * is
applied. */
        unsigned int    ping_number;            /* Ping sequenc
```

```
e number */
        float           sample_rate;           /* Sampling fre
quency in Hz for the Water Column Information */
        size_t          wcdata_alloc;          /* Number of po
ints allocated for wcdata arrays */
        float           *wcdata_x;             /* Distance in
meters to water column point port/stbd
                                                           * fro
m vessels heading. Negative value is port. */
        float           *wcdata_y;             /* Depth in met
ers for the water column point. */
        float           *wcdata_mag;           /* Intensity va
lue for water column point, not referenced */
        unsigned int    checksum;              /* checksum for
mat TBD */
       };
```

Step 4(a): Write the Initialization Functions in mbr_wasspunl.c

As noted above, all of the code to read and write the data format MBF_WASSPUNL should be located in the source file mbr_wasspunl.c. The functions that must be present are:

```
int mbr_register_wasspenl(int verbose, void *mbio_ptr,
        int *error);
int mbr_info_wasspenl(int verbose,
            int *system,
            int *beams_bath_max,
            int *beams_amp_max,
            int *pixels_ss_max,
            char *format_name,
            char *system_name,
            char *format_description,
            int *numfile,
            int *filetype,
            int *variable_beams,
            int *traveltime,
            int *beam_flagging,
            int *nav_source,
            int *heading_source,
            int *vru_source,
            int *svp_source,
            double *beamwidth_xtrack,
            double *beamwidth_ltrack,
            int *error);
int mbr_alm_wasspenl(int verbose, void *mbio_ptr, int *error);
int mbr_dem_wasspenl(int verbose, void *mbio_ptr, int *error);
int mbr_rt_wasspenl(int verbose, void *mbio_ptr, void *store_p
tr, int *error);
int mbr_wt_wasspenl(int verbose, void *mbio_ptr, void *store_p
tr, int *error);
```

The first three functions (*mbr_register_wasspenl()*, *mbr_info_wasspenl()*, and *mbr_alm_wasspenl()*) are part of the initialization following a call to *mb_read_init()* and *mb_write_init()*, and *mbr_dem_wasspenl()* is called by *mb_close()*. The reading and writing of files in the MBF_WASSPENL format is accomplished by *mbr_rt_wasspenl()* and *mbr_rt_wasspenl()*, respectively.

The function *mbr_register_wasspenl()* loads pointers to the data access functions in both mbr_wasspenl.c and mbsys_wassp.c into the MBIO data structure, allowing the proper I/O module functions to be called when high level MBIO

function calls are made. For instance, a call to *mb_extract()* by program mbprocess should result in a call to *mbsys_wassp_extract()* when the MBF_WASSPENL format has been initialized. Similarly, a call to *mb_read_ping()*, whether directly or through *mb_read()* or *mb_get_all()*, should result in a call to *mbr_rt_wasspenl()*. The code within function *mbr_register_wasspenl()* that accomplishes this registration looks like:

```
    /* set format and system specific function pointers */
    mb_io_ptr->mb_io_format_alloc = &mbr_alm_wasspenl;
    mb_io_ptr->mb_io_format_free = &mbr_dem_wasspenl;
    mb_io_ptr->mb_io_store_alloc = &mbsys_wassp_alloc;
    mb_io_ptr->mb_io_store_free = &mbsys_wassp_deall;
    mb_io_ptr->mb_io_read_ping = &mbr_rt_wasspenl;
    mb_io_ptr->mb_io_write_ping = &mbr_wt_wasspenl;
    mb_io_ptr->mb_io_dimensions = &mbsys_wassp_dimensions;
    mb_io_ptr->mb_io_pingnumber = &mbsys_wassp_pingnumber;
    mb_io_ptr->mb_io_sonartype = &mbsys_wassp_sonartype;
    mb_io_ptr->mb_io_sidescantype = NULL;
    mb_io_ptr->mb_io_extract = &mbsys_wassp_extract;
    mb_io_ptr->mb_io_insert = &mbsys_wassp_insert;
    mb_io_ptr->mb_io_extract_nav = &mbsys_wassp_extract_nav;
    mb_io_ptr->mb_io_extract_nnav = NULL;
    mb_io_ptr->mb_io_insert_nav = &mbsys_wassp_insert_nav;
    mb_io_ptr->mb_io_extract_altitude = &mbsys_wassp_extract_a
ltitude;
    mb_io_ptr->mb_io_insert_altitude = NULL;
    mb_io_ptr->mb_io_extract_svp = NULL;
    mb_io_ptr->mb_io_insert_svp = NULL;
    mb_io_ptr->mb_io_ttimes = &mbsys_wassp_ttimes;
    mb_io_ptr->mb_io_detects = &mbsys_wassp_detects;
    mb_io_ptr->mb_io_gains = &mbsys_wassp_gains;
    mb_io_ptr->mb_io_copyrecord = &mbsys_wassp_copy;
    mb_io_ptr->mb_io_extract_rawss = NULL;
    mb_io_ptr->mb_io_insert_rawss = NULL;
    mb_io_ptr->mb_io_extract_segytraceheader = NULL;
    mb_io_ptr->mb_io_extract_segy = NULL;
    mb_io_ptr->mb_io_insert_segy = NULL;
    mb_io_ptr->mb_io_ctd = NULL;
    mb_io_ptr->mb_io_ancilliarysensor = NULL;
```

Note that not all of the possible functions are registered. Only functions that are either required or sensible for this particular data format should be defined in mbsys_wassp.h, coded in mbsys_wassp.c, and registered in *mbr_register_wasspenl()*.

The function *mbr_info_wasspenl()* sets parameters and modes associated with

the data format, again as part of the initialization of reading or writing a file. The key code looks like:

```
    /* set format info parameters */
    status = MB_SUCCESS;
    *error = MB_ERROR_NO_ERROR;
    *system = MB_SYS_WASSP;
    *beams_bath_max = MBSYS_WASSP_MAX_BEAMS;
    *beams_amp_max = MBSYS_WASSP_MAX_BEAMS;
    *pixels_ss_max = MBSYS_WASSP_MAX_PIXELS;
    strncpy(format_name, "WASSPENL", MB_NAME_LENGTH);
    strncpy(system_name, "WASSP", MB_NAME_LENGTH);
    strncpy(format_description, "Format name:          MBF_WAS
SPENL\nInformal Description: WASSP Multibeam Vendor Format\nAt
tributes:           WASSP multibeams, \n
bathymetry and amplitude,\n             122 or 244 beams, bina
ry, Electronic Navigation Ltd.\n", MB_DESCRIPTION_LENGTH);
    *numfile = 1;
    *filetype = MB_FILETYPE_SINGLE;
    *variable_beams = MB_YES;
    *traveltime = MB_YES;
    *beam_flagging = MB_YES;
    *nav_source = MB_DATA_DATA;
    *heading_source = MB_DATA_DATA;
    *vru_source = MB_DATA_DATA;
    *svp_source = MB_DATA_NONE;
    *beamwidth_xtrack = 4.0;
    *beamwidth_ltrack = 4.0;
```

Here some of the values are self-explanatory, but others reflect the great variety is seafloor mapping data formats. The "MBSYS_WASSP_*" macros are declared in mbsys_wassp.h.

The *numfile* value is 1 for all formats storing data in single files, but can be 2 or 3 for formats with multiple parallel files. The *filetype* value identifies what i/o mechanism is used to read and files in this format; this must correspond to the actual reading and writing functions that are used in the functions in source file mbr_wasspenl.c. The possible values are:

```
/* types of  files used by swath sonar data formats */
#define MB_FILETYPE_NORMAL  1
#define MB_FILETYPE_SINGLE  2
#define MB_FILETYPE_XDR     3
#define MB_FILETYPE_GSF     4
#define MB_FILETYPE_NETCDF  5
#define MB_FILETYPE_SURF    6
#define MB_FILETYPE_SEGY    7
```

Files accessed through simple *fopen()*, *fread()*, *fwrite()*, and *fclose() system calls are "normal". Files accessed using the* mb_fileio_open()*,* mb_fileio_read()*,* mb_fileio_write()*, and* mb_fileio_close() *functions are "single", and support optimization of the data block buffer size through program mbdefaults. The other possible values denote formats accessed through external i/o libraries such as XDR, GSF, netCDF, SURF, and SEGY. Since the* filetype *value is MB_FILETYPE_SINGLE here, the* mb_fileio_*()* family of functions will be used in mbr_wasspenl.c.

For this format the number for beams can vary from ping to ping, and so *variable_beams* is true. The *traveltime* variable being true indicates that raw travel times (ranges) and angles are available, allowing recalcuation of bathymetry by raytracing.

The *nav_source*, *heading_source*, *vru_source*, and *svp_source* values indicate whether the primary source of these ancilliary data types are the survey records (MB_DATA_DATA), or other types of data records (defined in mbsystem/src/mbio/mb_status.h). Each of these values should be MB_DATA_DATA unless the information is only available in asynchronous records for this format. Values that are purely asynchronous (e.g. heading only available in MB_DATA_HEADING records) will need to stored and interpolated onto survey ping times within the *mbr_wasspenl_rt()* function using the *mb_navint_*()* family of functions.

The *beamwidth_xtrack* and *beamwidth_ltrack* values are the default acrosstrack and alongtrack beam widths for sounding in this format. In general, these values should be reset for each new ping in the I/O module. Future versions of MBIO will

include a more robust approach to defining beam widths of soundings.

Step 4(b): Write the Reading and Writing Functions in mbr_wasspunl.c

Basic Behavior of mbr_rt*() and mbr_wt*() Functions

The basic format-specific read and write functions for format MBF_WASSPENL are *mbr_rt_wasspenl()* and *mbr_wt_wasspenl()*, respectively. The full prototypes for these functions are:

```
int mbr_rt_wasspenl(int verbose, void *mbio_ptr,
                        void *store_ptr, int *error);
int mbr_wt_wasspenl(int verbose, void *mbio_ptr,
                        void *store_ptr, int *error);
```

where

- verbose ------- verbosity
- mbio_ptr ------ point to mb_io structure defining the reading or writing of a file through MBIO
- store_ptr ----- pointer to the primary data structure (usually also available in mb_io structure)
- error ---------- pointer to error value

The mbio_ptr and store_ptr pointers are obtained through mb_read_init() or mb_write_init(). The verbose value is set by the MB-System program, and the error value is used to return specific error states.

The behavior of *mbr_rt_wasspenl()* is to attempt to read the next data record from the open file referenced by mbio_ptr into the data structure referenced by store_ptr. The *status* and **error* return values indicate success or failure, and the *store->kind* value in the data structure indicates what type of record has been read. The return values are defined as follows:

- If a survey record is successfully read:

```
status=MB_SUCCESS;
*error=MB_ERROR_NO_ERROR;
store->kind = MB_DATA_DATA;
```

- If a comment record is successfully read:

```
status=MB_SUCCESS;
*error=MB_ERROR_NO_ERROR;
store->kind = MB_DATA_COMMENT;
```

- If some other record type is read:

```
status=MB_SUCCESS;
*error=MB_ERROR_NO_ERROR;
store->kind = MB_DATA_*;
```

- If the read fails:

```
status=MB_FAILURE;
*error=MB_ERROR_EOF;
store->kind = MB_DATA_NONE;
```

At the I/O module level in MBIO, any successful reading of a data record results in a *status* of MB_SUCCESS and an **error* of MB_ERROR_NO_ERROR. Higher level functions like *mb_get_all()* or *mb_read()* only return success for survey data (*store->kind* == MB_DATA_DATA), and will set nonfatal error conditions for comments, asynchronous navigation, and other sorts of data records.

The behavior of *mbr_wt_wasspenl()* is to attempt to write the current data record in the data structure referenced by store_ptr to the open file referenced by mbio_ptr. the *store->kind* value in the data structure indicates what type of record will be written. The *status* and **error* return values indicate success or failure.

The Structure of mb_rt_*() Functions: Simple Reading vs Multiple Functions

If the data format consists of one or two record types, then the reading and writing of those data can sensibly be coded directly in the *mb_rt_*()* function, and no other functions need be written or called. However, if the format is more complicated, with many data record types and possible variability in the type and order of records encountered in any particular file, then the format-specific code should be broken into many functions. The structure used for the writing functions should mirror that used for reading. The architecture used for reading and writing a particular format is always the choice of the developer. The only specific requirements are that the code work and be maintainable.

Reading by *mbr_rt_wasspenl()* and Subordinate Functions

For the MBF_WASSPENL format, we keep *mbr_rt_wasspenl()* quite simple, and architect the reading with two lower levels of functions. The function mbr_wasspenl_rd_data actually reads the file, identifying the start and size of records, and reading those records into a buffer. It then calls one of several functions to parse the specific record type and store the values in the data structure.

Here is the heart of *mbr_rt_wasspenl()*:

```
    /* get pointers to mbio descriptor */
    mb_io_ptr = (struct mb_io_struct *) mbio_ptr;

    /* read next data from file */
    status = mbr_wasspenl_rd_data(verbose,mbio_ptr,
                                  store_ptr,error);

    /* get pointers to data structures */
    store = (struct mbsys_wassp_struct *) store_ptr;

    /* set error and kind in mb_io_ptr */
    mb_io_ptr->new_error = *error;
    mb_io_ptr->new_kind = store->kind;
```

The function *mbr_wasspenl_rd_data()* is a great deal more complicated. First, it must find the start of the next record. The MBF_WASSPENL format data records

all begin with a similar first 16 bytes of the form:

```
unsigned int sync; // As a little endian unsigned int
                   //              value: 0x77F9345A
                   // The raw byte order is: 5A 34 F9 77
unsigned int size;  // Number of bytes in record from start
                   // of sync to end of checksum
char name[8];       // Record name - 8 characters,
                   // not null terminated
                   // Possible values: "GENBATHY",
                   // "CORBATHY", "RAWSONAR",
                   // "GEN_SENS", "NVUPDATE",
                   // "WCD_NAVI", "SYS_CFG1", "MCOMMENT"
```

To find the start of a valid record, the code reads the next 16 bytes into the start of a buffer, and then checks if the first four bytes match the expected sync value. If not, the code repeatedly drops the first byte, shifts the remaining 15 bytes over, and checks again until the sync value is found. The second four bytes give the size of the record; the code reads the rest of the record into the buffer starting at byte 16.

Note that the reading is done with *mb_fileio_get()* rather than *fread()*. This is because the *mb_io_ptr->filetype* parameter has been set to MB_FILETYPE_SINGLE, and allows the potential for optimizing system settings for file i/o.

Here is the code fragment that finds and reads the next record in the buffer:

```
        /* read next record header into buffer */
        read_len = (size_t)16;
        status = mb_fileio_get(verbose, mbio_ptr,
                               buffer, &read_len, error);

        /* check header - if not a good header read a byte
            at a time until a good header is found */
        skip = 0;
        while (status == MB_SUCCESS
            && *synctest != MBSYS_WASSP_SYNC)
```

```
            {
            /* get next byte */
            for (i=0;i<15;i++)
                buffer[i] = buffer[i+1];
            read_len = (size_t)1;
            status = mb_fileio_get(verbose, mbio_ptr,
                                   &buffer[15],
                                   &read_len, error);
            skip++;
            }

        /* get record id string */
        memcpy((void *)recordid, (void *)&buffer[8], (size_t)8
);
        recordid[9] = '\0';

        /* allocate memory to read rest of record if necessary
 */
        if (*bufferalloc < *record_size)
            {
            status = mb_reallocd(verbose, __FILE__, __LINE__,
                                   *record_size,
                                   (void **)bufferptr, error)
;
            if (status != MB_SUCCESS)
                {
                *bufferalloc = 0;
                done = MB_YES;
                }
            else
                {
                *bufferalloc = *record_size;
                buffer = (char *) *bufferptr;
                }
            }

        /* read the rest of the record */
        if (status == MB_SUCCESS)
            {
            read_len = (size_t)(*record_size - 16);
```

```
            status = mb_fileio_get(verbose, mbio_ptr,
                                   &buffer[16],
                                   &read_len, error);
            }
```

As can be seen above, the allocation of memory for the buffer is increased as necessary to hold the new record. Both the buffer pointer and the amount of memory allocated to it are stored in the mb_io_struct structure. There are number of integer, double, and pointer values in the mb_io_struct that are available for use by I/O modules. Here are example variable declarations and assignments:

```
char    **bufferptr;
char    *buffer;
int *bufferalloc;
unsigned int *synctest;
unsigned int *record_size;

bufferptr = (char **) &mb_io_ptr->saveptr1;
buffer = (char *) *bufferptr;
bufferalloc = (int *) &mb_io_ptr->save6;
synctest = (unsigned int *) buffer;
record_size = (unsigned int *)&buffer[4];
```

Since a sonar ping produces both the GENBATHY and CORBATHY records (and possibly the RAWSONAR record), the code must only return successful survey data when both GENBATHY and CORBATHY records have been read from the same ping. The GENBATHY record seems to come first, so the code never sets *done* true for GENBATHY records, and only checks for a full ping for CORBATHY records. All other records are assumed to stand alone (even the RAWSONAR), and so *done* is set true when they are read. The basic overall reading loop has the following structure:

```
    /* loop over reading data until a record is ready for retu
rn */
    done = MB_NO;
    *error = MB_ERROR_NO_ERROR;
    while (done == MB_NO)
```

```
        {
        // Code here to read the next record into a byte buffe
r called "buffer"

        /* if valid parse the record */
        if (status == MB_SUCCESS)
            {
            /* read GENBATHY record */
            if (strncmp(recordid, "GENBATHY", 8) == 0)
                {
                status = mbr_wasspenl_rd_genbathy(verbose, buf
fer, store_ptr, error);
                }

            /* read CORBATHY record */
            else if (strncmp(recordid, "CORBATHY", 8) == 0)
                {
                status = mbr_wasspenl_rd_corbathy(verbose, buf
fer, store_ptr, error);
                if (status == MB_SUCCESS)
                    {
                    if (genbathy->ping_number == corbathy->pin
g_number)
                        done = MB_YES;
                    else
                        {
                        status = MB_FAILURE;
                        *error = MB_ERROR_UNINTELLIGIBLE;
                        }
                    }
                }

            /* read RAWSONAR record */
            else if (strncmp(recordid, "RAWSONAR", 8) == 0)
                {
                status = mbr_wasspenl_rd_rawsonar(verbose, buf
fer, store_ptr, error);
                if (status == MB_SUCCESS)
                    done = MB_YES;
                }
```

```
            // Code to read other record types here....

            /* done if read success or EOF */
            if (status == MB_SUCCESS)
                {
                done = MB_YES;
                }
            else if (*error == MB_ERROR_EOF)
                {
                done = MB_YES;
                }
            }

        /* set done if read failure */
        else
            {
            done = MB_YES;
            }
        }
```

The functions that are used to parse the various data records after they are read are:

- *mbr_wasspenl_rd_genbathy()*
- *mbr_wasspenl_rd_corbathy()*
- *mbr_wasspenl_rd_rawsonar()*
- *mbr_wasspenl_rd_gen_sens()*
- *mbr_wasspenl_rd_nvupdate()*
- *mbr_wasspenl_rd_wcd_navi()*
- *mbr_wasspenl_rd_sys_cfg1()*
- *mbr_wasspenl_rd_mcomment()*
- *mbr_wasspenl_rd_unknown1()*

These functions follow a similar structure in which individual values are extracted from the buffer using functions like *mb_get_binary_int()*. These functions also set the kind and timestamp values in the data structure before returning. The function to parse GENBATHY records is shown here:

```
int mbr_wasspenl_rd_genbathy(int verbose, char *buffer, void *
store_ptr, int *error)
{
    char    *function_name = "mbr_wasspenl_rd_genbathy";
    int status = MB_SUCCESS;
    struct mbsys_wassp_struct *store;
    struct mbsys_wassp_genbathy_struct *genbathy;
    int index;
    int i;

    /* print input debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> called\n",f
unction_name);
        fprintf(stderr,"dbg2  Revision id: %s\n",rcs_id);
        fprintf(stderr,"dbg2  Input arguments:\n");
        fprintf(stderr,"dbg2       verbose:    %d\n",verbose);
        fprintf(stderr,"dbg2       buffer:     %p\n",(void *)b
uffer);
        fprintf(stderr,"dbg2       store_ptr:  %p\n",(void *)s
tore_ptr);
        }

    /* get pointer to raw data structure */
    store = (struct mbsys_wassp_struct *) store_ptr;
    genbathy = &(store->genbathy);

    /* extract the data */
    index = 16;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->vers
ion)); index += 4;
    mb_get_binary_double(MB_YES, &buffer[index], &(genbathy->m
sec)); index += 8;
    genbathy->day = buffer[index]; index++;
    genbathy->month = buffer[index]; index++;
    mb_get_binary_short(MB_YES, &buffer[index], &(genbathy->ye
ar)); index += 2;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->ping
```

```
_number)); index += 4;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->sona
r_model)); index += 4;
    mb_get_binary_long(MB_YES, &buffer[index], &(genbathy->tra
nsducer_serial_number)); index += 8;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->numb
er_beams)); index += 4;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->mode
flags)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->sa
mpling_frequency)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->ac
oustic_frequency)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->tx
_power)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->pu
lse_width)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->ab
sorption_loss)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->sp
reading_loss)); index += 4;
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->samp
le_type)); index += 4;
    mb_get_binary_float(MB_YES, &buffer[index], &(genbathy->so
und_velocity)); index += 4;
    for (i=0;i<genbathy->number_beams;i++)
        {
        mb_get_binary_float(MB_YES, &buffer[index], &(genbathy
->detection_point[i])); index += 4;
        mb_get_binary_float(MB_YES, &buffer[index], &(genbathy
->rx_angle[i])); index += 4;
        mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->
flags[i])); index += 4;
        mb_get_binary_float(MB_YES, &buffer[index], &(genbathy
->backscatter[i])); index += 4;
        }
    mb_get_binary_int(MB_YES, &buffer[index], &(genbathy->chec
ksum)); index += 4;

    /* set kind */
```

```
    if (status == MB_SUCCESS)
        {
        /* set kind */
        store->kind = MB_DATA_DATA;

        /* get the time */
        store->time_i[0] = genbathy->year;
        store->time_i[1] = genbathy->month;
        store->time_i[2] = genbathy->day;
        store->time_i[3] = (int)floor(genbathy->msec / 3600000
.0);
        store->time_i[4] = (int)floor((genbathy->msec - 360000
0.0 * store->time_i[3]) / 60000.0);
        store->time_i[5] = (int)floor((genbathy->msec
                            - 3600000.0 * store->time_i[3]
                            - 60000.0 * store->time_i[4]) / 100
0.0);;
        store->time_i[6] = (int)((genbathy->msec
                            - 3600000.0 * store->time_i[3]
                            - 60000.0 * store->time_i[4]
                            - 1000.0 * store->time_i[5]) * 1000
.0);;
        mb_get_time(verbose, store->time_i, &(store->time_d));
        }
    else
        {
        store->kind = MB_DATA_NONE;
        }

    /* print debug statements */
    if (verbose >= 5)
        {
        fprintf(stderr,"\ndbg5  Values read in MBIO function <
%s>\n",function_name);
        fprintf(stderr,"dbg5       genbathy->version:
           %u\n",genbathy->version);
        fprintf(stderr,"dbg5       genbathy->msec:
           %f\n",genbathy->msec);
        fprintf(stderr,"dbg5       genbathy->day:
           %u\n",genbathy->day);
```

```
        fprintf(stderr,"dbg5       genbathy->month:
           %u\n",genbathy->month);
        fprintf(stderr,"dbg5       genbathy->year:
           %u\n",genbathy->year);
        fprintf(stderr,"dbg5       genbathy->ping_number:
           %u\n",genbathy->ping_number);
        fprintf(stderr,"dbg5       genbathy->sonar_model:
           %u\n",genbathy->sonar_model);
        fprintf(stderr,"dbg5       genbathy->transducer_serial
_number:   %lu\n",genbathy->transducer_serial_number);
        fprintf(stderr,"dbg5       genbathy->number_beams:
           %u\n",genbathy->number_beams);
        fprintf(stderr,"dbg5       genbathy->modeflags:
           %u\n",genbathy->modeflags);
        fprintf(stderr,"dbg5       genbathy->sampling_frequenc
y:         %f\n",genbathy->sampling_frequency);
        fprintf(stderr,"dbg5       genbathy->acoustic_frequenc
y:         %f\n",genbathy->acoustic_frequency);
        fprintf(stderr,"dbg5       genbathy->tx_power:
           %f\n",genbathy->tx_power);
        fprintf(stderr,"dbg5       genbathy->pulse_width:
           %f\n",genbathy->pulse_width);
        fprintf(stderr,"dbg5       genbathy->absorption_loss:
           %f\n",genbathy->absorption_loss);
        fprintf(stderr,"dbg5       genbathy->spreading_loss:
           %f\n",genbathy->spreading_loss);
        fprintf(stderr,"dbg5       genbathy->sample_type:
           %u\n",genbathy->sample_type);
        fprintf(stderr,"dbg5       genbathy->sound_velocity:
           %f\n",genbathy->sound_velocity);
        for (i=0;i<genbathy->number_beams;i++)
            {
            fprintf(stderr,"dbg5       genbathy->detection_poi
nt[%3d]:       %f\n",i,genbathy->detection_point[i]);
            fprintf(stderr,"dbg5       genbathy->rx_angle[%3d]
:             %f\n",i,genbathy->rx_angle[i]);
            fprintf(stderr,"dbg5       genbathy->flags[%3d]:
               %u\n",i,genbathy->flags[i]);
            fprintf(stderr,"dbg5       genbathy->backscatter[%
3d]:           %f\n",i,genbathy->backscatter[i]);
```

```
            }
        fprintf(stderr,"dbg5       genbathy->checksum:
           %u\n",genbathy->checksum);
        }

    /* print output debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> completed\n
",function_name);
        fprintf(stderr,"dbg2  Return values:\n");
        fprintf(stderr,"dbg2       error:      %d\n",*error);
        fprintf(stderr,"dbg2  Return status:\n");
        fprintf(stderr,"dbg2       status:  %d\n",status);
        }

    /* return status */
    return(status);
}
```

The mbr_wasspenl_rd_genbathy() function also illustrates a couple of MB-System coding conventions. First, all MB-System functions should print out the entry and return values if *verbose* >= 2 to stderr. Second, if *verbose* >= 5, all functions that read or write data records should print out all of the values being read or written.

On-The-Fly Interpolation of Asynchronous Values (Not Needed for WASSP)

In the case of data formats that have no navigation, heading, sensor depth, or attitude data stored in the survey records, these values must be obtained by interpolation of values found in asynchronous records. For the WASSP data, this is not an issue. If on-the-fly interpolation of the ancilliary values is needed, the general logic that should be placed in the *mbr_rt_*()* function is something this:

```
// This example is for general illustration only!!!
// Also, in real I/O modules the key parameters are never simp
ly
// stored as store->longitude, store->heading, etc...
```

```
// This is where the code goes to read the next data record
// In mbr_rt_wasspenl() this is the call to mb_wasspenl_rd_dat
a()

// Add asynchronous values to the interpolation arrays if
// this is the right kind of data record
if (mb_io_ptr->nav_source != MB_DATA_DATA
    && (store->kind == mb_io_ptr->nav_source
        || store->kind == mb_io_ptr->heading_source
        || store->kind == mb_io_ptr->vru_source))
    {
    status = mb_extract_nav(verbose, mbio_ptr, *store_ptr,
                                &kind, time_i, &time_d,
                                &longitude, &latitude, &speed,
 &heading,
                                &sensordepth, &roll, &pitch, &
heave,
                                error);
    if (store->kind == mb_io_ptr->nav_source)
        {
        status = mb_navint_add(verbose, mbio_ptr, store->time_
d,
                                    longitude, latitude, error
);
        status = mb_depint_add(verbose, mbio_ptr, store->time_
d,
                                    sensordepth, error);
        }
    if (store->kind == mb_io_ptr->heading_source)
        status = mb_hedint_add(verbose, mbio_ptr, store->time_
d,
                                    heading, error);
    if (store->kind == mb_io_ptr->vru_source)
        status = mb_attint_add(verbose, mbio_ptr, store->time_
d,
                                    roll, pitch, heave, error)
;
    }
```

```
// If this is a survey record then interpolate ancilliary valu
es
if (store->kind == MB_DATA_DATA)
    {
    interp_status = mb_hedint_interp(verbose, mbio_ptr, store-
>time_d,
                        &store->heading, &interp_error);
    interp_status = mb_navint_interp(verbose, mbio_ptr, store-
>time_d, heading, 0.0,
                        &store->longitude, &store->latitude, &
store->speed, &interp_error);
    interp_status = mb_depint_interp(verbose, mbio_ptr, store-
>time_d,
                        &store->sensordepth, &interp_error);
    interp_status = mb_attint_interp(verbose, mbio_ptr, store-
>time_d,
                        &store->heave, &store->roll, &store->p
itch, &interp_error);
    }
```

The above code is for illustration only, this is not actual code from any of the I/O modules. In most of the cases where on-the-fly interpolation is needed, there are multiple possible types of asynchronous records, and so the actual code is more complicated. Also, the actual storage variables are rarely as simple as "store->heading".

Writing by *mbr_wt_wasspenl()* and Subordinate Functions

The *mbr_wt_wasspenl()* function is similar to *mbr_rt_wasspenl()* in that it does little more than call *mbr_wasspenl_wr_data()*:

```
    /* get pointer to mbio descriptor */
    mb_io_ptr = (struct mb_io_struct *) mbio_ptr;

    /* get pointer to raw data structure */
    store = (struct mbsys_wassp_struct *) store_ptr;

    /* write next data to file */
    status = mbr_wasspenl_wr_data(verbose,mbio_ptr,store_ptr,e
rror);
```

The function *mbr_wasspenl_wr_data()* in turn simply calls functions to insert the desired output data record into the buffer, and then writes that buffer to the output file referenced by *mbio_ptr*. In the case of survey data, both GENBATHY and CORBATHY records are written out.

```
    /* get pointer to mbio descriptor */
    mb_io_ptr = (struct mb_io_struct *) mbio_ptr;

    /* get pointer to raw data structure */
    store = (struct mbsys_wassp_struct *) store_ptr;

    /* get saved values */
    bufferptr = (char **) &mb_io_ptr->saveptr1;
    buffer = (char *) *bufferptr;
    bufferalloc = (int *) &mb_io_ptr->save6;

    /* write the current data record */

    /* write GENBATHY record */
    if (store->kind == MB_DATA_DATA)
        {
        status = mbr_wasspenl_wr_genbathy(verbose, bufferalloc
, bufferptr, store_ptr, &size, error);
        buffer = (char *) *bufferptr;
        write_len = (size_t)size;
        status = mb_fileio_put(verbose, mbio_ptr, buffer, &wri
te_len, error);
```

```
        status = mbr_wasspenl_wr_corbathy(verbose, bufferalloc
, bufferptr, store_ptr, &size, error);
        buffer = (char *) *bufferptr;
        write_len = (size_t)size;
        status = mb_fileio_put(verbose, mbio_ptr, buffer, &wri
te_len, error);
        }

    /* write RAWSONAR record */
    else if (store->kind == MB_DATA_WATER_COLUMN)
        {
        status = mbr_wasspenl_wr_rawsonar(verbose, bufferalloc
, bufferptr, store_ptr, &size, error);
        buffer = (char *) *bufferptr;
        write_len = (size_t)size;
        status = mb_fileio_put(verbose, mbio_ptr, buffer, &wri
te_len, error);
        }

    // more code to write all the other record types....
```

The functions that are used to construct the various data records before they are written are:

- *mbr_wasspenl_wr_genbathy()*
- *mbr_wasspenl_wr_corbathy()*
- *mbr_wasspenl_wr_rawsonar()*
- *mbr_wasspenl_wr_gen_sens()*
- *mbr_wasspenl_wr_nvupdate()*
- *mbr_wasspenl_wr_wcd_navi()*
- *mbr_wasspenl_wr_sys_cfg1()*
- *mbr_wasspenl_wr_mcomment()*
- *mbr_wasspenl_wr_unknown1()*

These functions follow a similar structure in which individual values are inserted into the buffer using functions like *mb_put_binary_int()*. The function to construct GENBATHY records is shown here:

```
int mbr_wasspenl_wr_genbathy(int verbose, int *bufferalloc, ch
ar **bufferptr, void *store_ptr, int *size, int *error)
{
    char    *function_name = "mbr_wasspenl_wr_genbathy";
    int status = MB_SUCCESS;
    struct mbsys_wassp_struct *store;
    struct mbsys_wassp_genbathy_struct *genbathy;
    char    *buffer;
    int index;
    int i;

    /* print input debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> called\n",f
unction_name);
        fprintf(stderr,"dbg2  Revision id: %s\n",rcs_id);
        fprintf(stderr,"dbg2  Input arguments:\n");
        fprintf(stderr,"dbg2       verbose:    %d\n",verbose);
        fprintf(stderr,"dbg2       bufferalloc:%d\n",*bufferal
loc);
        fprintf(stderr,"dbg2       bufferptr:  %p\n",(void *)b
ufferptr);
        fprintf(stderr,"dbg2       store_ptr:  %p\n",(void *)s
tore_ptr);
        }

    /* get pointer to raw data structure */
    store = (struct mbsys_wassp_struct *) store_ptr;
    genbathy = &(store->genbathy);

    /* print debug statements */
    if (verbose >= 5)
        {
        fprintf(stderr,"\ndbg5  Values to be written in MBIO f
unction <%s>\n",function_name);
        fprintf(stderr,"dbg5       genbathy->version:
           %u\n",genbathy->version);
        fprintf(stderr,"dbg5       genbathy->msec:
```

```
            %f\n",genbathy->msec);
        fprintf(stderr,"dbg5       genbathy->day:
            %u\n",genbathy->day);
        fprintf(stderr,"dbg5       genbathy->month:
            %u\n",genbathy->month);
        fprintf(stderr,"dbg5       genbathy->year:
            %u\n",genbathy->year);
        fprintf(stderr,"dbg5       genbathy->ping_number:
            %u\n",genbathy->ping_number);
        fprintf(stderr,"dbg5       genbathy->sonar_model:
            %u\n",genbathy->sonar_model);
        fprintf(stderr,"dbg5       genbathy->transducer_serial
_number:   %lu\n",genbathy->transducer_serial_number);
        fprintf(stderr,"dbg5       genbathy->number_beams:
            %u\n",genbathy->number_beams);
        fprintf(stderr,"dbg5       genbathy->modeflags:
            %u\n",genbathy->modeflags);
        fprintf(stderr,"dbg5       genbathy->sampling_frequenc
y:          %f\n",genbathy->sampling_frequency);
        fprintf(stderr,"dbg5       genbathy->acoustic_frequenc
y:          %f\n",genbathy->acoustic_frequency);
        fprintf(stderr,"dbg5       genbathy->tx_power:
            %f\n",genbathy->tx_power);
        fprintf(stderr,"dbg5       genbathy->pulse_width:
            %f\n",genbathy->pulse_width);
        fprintf(stderr,"dbg5       genbathy->absorption_loss:
            %f\n",genbathy->absorption_loss);
        fprintf(stderr,"dbg5       genbathy->spreading_loss:
            %f\n",genbathy->spreading_loss);
        fprintf(stderr,"dbg5       genbathy->sample_type:
            %u\n",genbathy->sample_type);
        fprintf(stderr,"dbg5       genbathy->sound_velocity:
            %f\n",genbathy->sound_velocity);
        for (i=0;i<genbathy->number_beams;i++)
            {
            fprintf(stderr,"dbg5       genbathy->detection_poi
nt[%3d]:       %f\n",i,genbathy->detection_point[i]);
            fprintf(stderr,"dbg5       genbathy->rx_angle[%3d]
:             %f\n",i,genbathy->rx_angle[i]);
            fprintf(stderr,"dbg5       genbathy->flags[%3d]:
```

```
                %u\n",i,genbathy->flags[i]);
            fprintf(stderr,"dbg5       genbathy->backscatter[%
3d]:           %f\n",i,genbathy->backscatter[i]);
            }
        fprintf(stderr,"dbg5       genbathy->checksum:
           %u\n",genbathy->checksum);
        }

    /* figure out size of output record */
    *size = 92 + 16 * genbathy->number_beams;

    /* allocate memory to write rest of record if necessary */
    if (*bufferalloc < *size)
        {
        status = mb_reallocd(verbose, __FILE__, __LINE__, *siz
e,
                    (void **)bufferptr, error);
        if (status != MB_SUCCESS)
            *bufferalloc = 0;
        else
            *bufferalloc = *size;
        }

    /* proceed to write if buffer allocated */
    if (status == MB_SUCCESS)
        {
        /* get buffer for writing */
        buffer = (char *) *bufferptr;

        /* insert the data */
        index = 0;
        mb_put_binary_int(MB_YES, MBSYS_WASSP_SYNC, &buffer[in
dex]); index += 4;
        mb_put_binary_int(MB_YES, *size, &buffer[index]); inde
x += 4;
        strncpy(&buffer[index], "GENBATHY", 8); index += 8;
        mb_put_binary_int(MB_YES, genbathy->version, &buffer[i
ndex]); index += 4;
        mb_put_binary_double(MB_YES, genbathy->msec, &buffer[i
ndex]); index += 8;
```

```
        buffer[index] = genbathy->day; index++;
        buffer[index] = genbathy->month; index++;
        mb_put_binary_short(MB_YES, genbathy->year, &buffer[in
dex]); index += 2;
        mb_put_binary_int(MB_YES, genbathy->ping_number, &buff
er[index]); index += 4;
        mb_put_binary_int(MB_YES, genbathy->sonar_model, &buff
er[index]); index += 4;
        mb_put_binary_long(MB_YES, genbathy->transducer_serial
_number, &buffer[index]); index += 8;
        mb_put_binary_int(MB_YES, genbathy->number_beams, &buf
fer[index]); index += 4;
        mb_put_binary_int(MB_YES, genbathy->modeflags, &buffer
[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->sampling_frequen
cy, &buffer[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->acoustic_frequen
cy, &buffer[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->tx_power, &buffe
r[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->pulse_width, &bu
ffer[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->absorption_loss,
 &buffer[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->spreading_loss,
&buffer[index]); index += 4;
        mb_put_binary_int(MB_YES, genbathy->sample_type, &buff
er[index]); index += 4;
        mb_put_binary_float(MB_YES, genbathy->sound_velocity,
&buffer[index]); index += 4;
        for (i=0;i<genbathy->number_beams;i++)
            {
            mb_put_binary_float(MB_YES, genbathy->detection_po
int[i], &buffer[index]); index += 4;
            mb_put_binary_float(MB_YES, genbathy->rx_angle[i],
 &buffer[index]); index += 4;
            mb_put_binary_int(MB_YES, genbathy->flags[i], &buf
fer[index]); index += 4;
            mb_put_binary_float(MB_YES, genbathy->backscatter[
i], &buffer[index]); index += 4;
```

```
            }

        /* now add the checksum */
        genbathy->checksum = 0;
        for (i=0;i<index;i++)
            genbathy->checksum += (unsigned char) buffer[i];
        mb_put_binary_int(MB_YES, genbathy->checksum, &buffer[
index]); index += 4;
        }

    /* print output debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> completed\n
",function_name);
        fprintf(stderr,"dbg2  Return values:\n");
        fprintf(stderr,"dbg2       error:      %d\n",*error);
        fprintf(stderr,"dbg2  Return status:\n");
        fprintf(stderr,"dbg2       status:  %d\n",status);
        }

    /* return status */
    return(status);
}
```

Step 5. Write the Data Access Functions in mbsys_wassp.c

As noted above, the source file mbsystem/src/mbio/mbsys_wassp.c must contain the code for all of the *mbsys_wassp_*()* functions loaded into the mb_io_struct I/O structure by the *mbr_register_wasspenl()* function. These are:

- *mbsys_wassp_dimensions()*
- *mbsys_wassp_pingnumber()*
- *mbsys_wassp_sonartype()*
- *mbsys_wassp_extract()*
- *mbsys_wassp_insert()*
- *mbsys_wassp_ttimes()*
- *mbsys_wassp_detects()*

- *mbsys_wassp_extract_nav()*
- *mbsys_wassp_insert_nav()*
- *mbsys_wassp_extract_altitude()*
- *mbsys_wassp_detects()*
- *mbsys_wassp_gains()*
- *mbsys_wassp_copyrecord()*

Each of these functions either extracts or inserts a well defined set of values from or to the data structure. Care must be taken to translate the values correctly between the units used in the data format and those used by the MBIO API. For instance, the longitude and latitude values are in decimal degrees, the sensor depth value is in meters, and the speed value is in km/hour.

One interesting complication with the WASSP format is that the sounding locations in the CORBATHY records are stored as distances east and north of the sensor navigation point. Most multibeam formats store these locations as distances acrosstrack-starboard and alongtrack-forward of the sensor, and that is certainly the form expected in the arrays returned by *mbsys_wassp_extract()*. The sounding positions must be translated between the two conventions in both *mbsys_wassp_extract()* and *mbsys_wassp_insert()*, a calculation that depends on the heading value. The translation from relative easting-northing to acrosstrack and alongtrack distances in *mbsys_wassp_extract()* is in this code fragment:

```
        /* get coordinate scaling */
        headingx = sin(-(*heading)*DTR);
        headingy = cos(-(*heading)*DTR);

        /* read distance and depth values into storage arrays
*/
        *nbath = corbathy->num_beams;
        *namp = *nbath;
        for (i=0;i<*nbath;i++)
            {
            bath[i] = 0.0;
            beamflag[i] = MB_FLAG_NULL;
            bathacrosstrack[i] = 0.0;
            bathalongtrack[i] = 0.0;
            amp[i] = 0.0;
            }
        for (i=0;i<*nbath;i++)
            {
            j = corbathy->beam_index[i];
            bath[j] = -corbathy->z[i];
            beamflag[j] = corbathy->empty[i];
            bathacrosstrack[j] = headingy * corbathy->x[i] + h
eadingx * (-corbathy->y[i]);
            bathalongtrack[j] = -headingx * corbathy->x[i] + h
eadingy * (-corbathy->y[i]);
            amp[j] = corbathy->backscatter[i];
            }
```

Here the *sin()* and *cos()* calculations include use of a preprocessor macro DTR. This stands for degrees-to-radians and is defined as M_PI / 180.0 in the header file mbsystem/src/mbio/mb_define.h (along with RTD, or radians-to-degrees). Also note that the WASSP beam data arrays include beam_index[], which is the actual beam number for each sounding. The use of index values means that some beams may not be specified and should presumably be treated as null. Consequently, each extraction must begin by initializing the target arrays to null, as is done in the first loop over *nbath.

Here is the full code for the most used data access function,

mbsys_wassp_extract():

```
int mbsys_wassp_extract(int verbose, void *mbio_ptr, void *sto
re_ptr,
        int *kind, int time_i[7], double *time_d,
        double *navlon, double *navlat,
        double *speed, double *heading,
        int *nbath, int *namp, int *nss,
        char *beamflag, double *bath, double *amp,
        double *bathacrosstrack, double *bathalongtrack,
        double *ss, double *ssacrosstrack, double *ssalongtrac
k,
        char *comment, int *error)
{
    char    *function_name = "mbsys_wassp_extract";
    int status = MB_SUCCESS;
    struct mb_io_struct *mb_io_ptr;
    struct mbsys_wassp_struct *store;
    struct mbsys_wassp_genbathy_struct *genbathy;
    struct mbsys_wassp_corbathy_struct *corbathy;
    struct mbsys_wassp_rawsonar_struct *rawsonar;
    struct mbsys_wassp_gen_sens_struct *gen_sens;
    struct mbsys_wassp_nvupdate_struct *nvupdate;
    struct mbsys_wassp_wcd_navi_struct *wcd_navi;
    struct mbsys_wassp_sys_cfg1_struct *sys_cfg1;
    struct mbsys_wassp_mcomment_struct *mcomment;
    double  headingx, headingy;
    double  dx, dy;
    int i, j;

    /* print input debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> called\n",f
unction_name);
        fprintf(stderr,"dbg2  Revision id: %s\n",version_id);
        fprintf(stderr,"dbg2  Input arguments:\n");
        fprintf(stderr,"dbg2       verbose:    %d\n",verbose);
        fprintf(stderr,"dbg2       mb_ptr:     %p\n",(void *)m
```

```
bio_ptr);
        fprintf(stderr,"dbg2       store_ptr:  %p\n",(void *)s
tore_ptr);
        }

    /* get mbio descriptor */
    mb_io_ptr = (struct mb_io_struct *) mbio_ptr;

    /* get data structure pointer */
    store = (struct mbsys_wassp_struct *) store_ptr;
    genbathy = (struct mbsys_wassp_genbathy_struct *) &(store-
>genbathy);
    corbathy = (struct mbsys_wassp_corbathy_struct *) &(store-
>corbathy);
    rawsonar = (struct mbsys_wassp_rawsonar_struct *) &(store-
>rawsonar);
    gen_sens = (struct mbsys_wassp_gen_sens_struct *) &(store-
>gen_sens);
    nvupdate = (struct mbsys_wassp_nvupdate_struct *) &(store-
>nvupdate);
    wcd_navi = (struct mbsys_wassp_wcd_navi_struct *) &(store-
>wcd_navi);
    sys_cfg1 = (struct mbsys_wassp_sys_cfg1_struct *) &(store-
>sys_cfg1);
    mcomment = (struct mbsys_wassp_mcomment_struct *) &(store-
>mcomment);

    /* get data kind */
    *kind = store->kind;

    /* extract data from structure */
    if (*kind == MB_DATA_DATA)
        {
        /* get time */
        for (i=0;i<7;i++)
            time_i[i] = store->time_i[i];
        *time_d = store->time_d;

        /* get navigation */
        *navlon = corbathy->longitude;
```

```
        *navlat = corbathy->latitude;

        /* get speed */
        *speed = 1.8520 * nvupdate->sog;

        /* get heading */
        *heading = corbathy->bearing;

        /* set beamwidths in mb_io structure */
        mb_io_ptr->beamwidth_xtrack = 4.0;
        mb_io_ptr->beamwidth_ltrack = 4.0;

        /* get coordinate scaling */
        headingx = sin(-(*heading)*DTR);
        headingy = cos(-(*heading)*DTR);

        /* read distance and depth values into storage arrays
*/
        *nbath = corbathy->num_beams;
        *namp = *nbath;
        for (i=0;i<*nbath;i++)
            {
            bath[i] = 0.0;
            beamflag[i] = MB_FLAG_NULL;
            bathacrosstrack[i] = 0.0;
            bathalongtrack[i] = 0.0;
            amp[i] = 0.0;
            }
        for (i=0;i<*nbath;i++)
            {
            j = corbathy->beam_index[i];
            bath[j] = -corbathy->z[i];
            beamflag[j] = corbathy->empty[i];
            bathacrosstrack[j] = headingy * corbathy->x[i] + h
eadingx * (-corbathy->y[i]);
            bathalongtrack[j] = -headingx * corbathy->x[i] + h
eadingy * (-corbathy->y[i]);
            amp[j] = corbathy->backscatter[i];
            }
```

```
        /* extract sidescan */
        *nss = 0;

        /* print debug statements */
        if (verbose >= 5)
            {
            fprintf(stderr,"\ndbg4  Data extracted by MBIO fun
ction <%s>\n",
                function_name);
            fprintf(stderr,"dbg4  Extracted values:\n");
            fprintf(stderr,"dbg4       kind:       %d\n", *kin
d);
            fprintf(stderr,"dbg4       error:      %d\n", *err
or);
            fprintf(stderr,"dbg4       time_i[0]:  %d\n", time
_i[0]);
            fprintf(stderr,"dbg4       time_i[1]:  %d\n", time
_i[1]);
            fprintf(stderr,"dbg4       time_i[2]:  %d\n", time
_i[2]);
            fprintf(stderr,"dbg4       time_i[3]:  %d\n", time
_i[3]);
            fprintf(stderr,"dbg4       time_i[4]:  %d\n", time
_i[4]);
            fprintf(stderr,"dbg4       time_i[5]:  %d\n", time
_i[5]);
            fprintf(stderr,"dbg4       time_i[6]:  %d\n", time
_i[6]);
            fprintf(stderr,"dbg4       time_d:     %f\n", *tim
e_d);
            fprintf(stderr,"dbg4       longitude:  %f\n", *nav
lon);
            fprintf(stderr,"dbg4       latitude:   %f\n", *nav
lat);
            fprintf(stderr,"dbg4       speed:      %f\n", *spe
ed);
            fprintf(stderr,"dbg4       heading:    %f\n", *hea
ding);
            fprintf(stderr,"dbg4       nbath:      %d\n", *nba
th);
```

```
            for (i=0;i<*nbath;i++)
              fprintf(stderr,"dbg4       beam:%d  flag:%3d  ba
th:%f  acrosstrack:%f  alongtrack:%f\n",
                i,beamflag[i],bath[i],
                bathacrosstrack[i],bathalongtrack[i]);
            fprintf(stderr,"dbg4        namp:     %d\n", *namp
);
            for (i=0;i<*namp;i++)
              fprintf(stderr,"dbg4        beam:%d   amp:%f  ac
rosstrack:%f  alongtrack:%f\n",
                i,amp[i],bathacrosstrack[i],bathalongtrack[i])
;
            fprintf(stderr,"dbg4        nss:      %d\n", *nss)
;
            for (i=0;i<*nss;i++)
              fprintf(stderr,"dbg4        pixel:%d   ss:%f  ac
rosstrack:%f  alongtrack:%f\n",
                i,ss[i],ssacrosstrack[i],ssalongtrack[i]);
            }

        /* done translating values */

        }

    /* extract data from structure */
    else if (*kind == MB_DATA_NAV)
        {
        /* get time */
        for (i=0;i<7;i++)
            time_i[i] = store->time_i[i];
        *time_d = store->time_d;

        /* get navigation */
        *navlon = nvupdate->longitude;
        *navlat = nvupdate->latitude;

        /* get speed */
        *speed = 1.8520 * nvupdate->sog;

        /* get heading */
```

```
        *heading = nvupdate->heading;

        /* set beam and pixel numbers */
        *nbath = 0;
        *namp = 0;
        *nss = 0;

        /* print debug statements */
        if (verbose >= 5)
            {
            fprintf(stderr,"\ndbg4  Data extracted by MBIO fun
ction <%s>\n",
                function_name);
            fprintf(stderr,"dbg4  Extracted values:\n");
            fprintf(stderr,"dbg4       kind:       %d\n", *kin
d);
            fprintf(stderr,"dbg4       error:      %d\n", *err
or);
            fprintf(stderr,"dbg4       time_i[0]:  %d\n", time
_i[0]);
            fprintf(stderr,"dbg4       time_i[1]:  %d\n", time
_i[1]);
            fprintf(stderr,"dbg4       time_i[2]:  %d\n", time
_i[2]);
            fprintf(stderr,"dbg4       time_i[3]:  %d\n", time
_i[3]);
            fprintf(stderr,"dbg4       time_i[4]:  %d\n", time
_i[4]);
            fprintf(stderr,"dbg4       time_i[5]:  %d\n", time
_i[5]);
            fprintf(stderr,"dbg4       time_i[6]:  %d\n", time
_i[6]);
            fprintf(stderr,"dbg4       time_d:     %f\n", *tim
e_d);
            fprintf(stderr,"dbg4       longitude:  %f\n", *nav
lon);
            fprintf(stderr,"dbg4       latitude:   %f\n", *nav
lat);
            fprintf(stderr,"dbg4       speed:      %f\n", *spe
ed);
```

```
            fprintf(stderr,"dbg4       heading:    %f\n", *hea
ding);
            }

        /* done translating values */

        }

    /* extract comment from structure */
    else if (*kind == MB_DATA_COMMENT)
        {
        /* get time */
        for (i=0;i<7;i++)
            time_i[i] = store->time_i[i];
        *time_d = store->time_d;

        /* copy comment */
        if (mcomment->comment_length > 0)
            strncpy(comment, mcomment->comment_message, MIN(mc
omment->comment_length, MB_COMMENT_MAXLINE));
        else
            comment[0] = '\0';

        /* print debug statements */
        if (verbose >= 4)
            {
            fprintf(stderr,"\ndbg4  Comment extracted by MBIO
function <%s>\n",
                function_name);
            fprintf(stderr,"dbg4  New ping values:\n");
            fprintf(stderr,"dbg4       kind:       %d\n", *kin
d);
            fprintf(stderr,"dbg4       error:      %d\n", *err
or);
            fprintf(stderr,"dbg4       time_i[0]:  %d\n", time
_i[0]);
            fprintf(stderr,"dbg4       time_i[1]:  %d\n", time
_i[1]);
            fprintf(stderr,"dbg4       time_i[2]:  %d\n", time
_i[2]);
```

```
            fprintf(stderr,"dbg4       time_i[3]:  %d\n", time
_i[3]);
            fprintf(stderr,"dbg4       time_i[4]:  %d\n", time
_i[4]);
            fprintf(stderr,"dbg4       time_i[5]:  %d\n", time
_i[5]);
            fprintf(stderr,"dbg4       time_i[6]:  %d\n", time
_i[6]);
            fprintf(stderr,"dbg4       time_d:     %f\n", *tim
e_d);
            fprintf(stderr,"dbg4       comment:    %s\n", comm
ent);
            }
        }

    /* set time for other data records */
    else
        {
        /* get time */
        for (i=0;i<7;i++)
            time_i[i] = store->time_i[i];
        *time_d = store->time_d;

        /* print debug statements */
        if (verbose >= 4)
            {
            fprintf(stderr,"\ndbg4  Data extracted by MBIO fun
ction <%s>\n",
                function_name);
            fprintf(stderr,"dbg4  Extracted values:\n");
            fprintf(stderr,"dbg4       kind:       %d\n",*kind
);
            fprintf(stderr,"dbg4       error:      %d\n",*erro
r);
            fprintf(stderr,"dbg4       time_i[0]:  %d\n",time_
i[0]);
            fprintf(stderr,"dbg4       time_i[1]:  %d\n",time_
i[1]);
            fprintf(stderr,"dbg4       time_i[2]:  %d\n",time_
i[2]);
```

```
            fprintf(stderr,"dbg4       time_i[3]:  %d\n",time_
i[3]);
            fprintf(stderr,"dbg4       time_i[4]:  %d\n",time_
i[4]);
            fprintf(stderr,"dbg4       time_i[5]:  %d\n",time_
i[5]);
            fprintf(stderr,"dbg4       time_i[6]:  %d\n",time_
i[6]);
            fprintf(stderr,"dbg4       time_d:     %f\n",*time
_d);
            fprintf(stderr,"dbg4       comment:    %s\n",comme
nt);
            }
        }

    /* print output debug statements */
    if (verbose >= 2)
        {
        fprintf(stderr,"\ndbg2  MBIO function <%s> completed\n
",function_name);
        fprintf(stderr,"dbg2  Return values:\n");
        fprintf(stderr,"dbg2       kind:       %d\n",*kind);
        }
    if (verbose >= 2 && *error <= MB_ERROR_NO_ERROR
        && *kind == MB_DATA_COMMENT)
        {
        fprintf(stderr,"dbg2       comment:     \ndbg2       %
s\n",
            comment);
        }
    else if (verbose >= 2 && *error <= MB_ERROR_NO_ERROR
        && *kind != MB_DATA_COMMENT)
        {
        fprintf(stderr,"dbg2       time_i[0]:     %d\n",time_i
[0]);
        fprintf(stderr,"dbg2       time_i[1]:     %d\n",time_i
[1]);
        fprintf(stderr,"dbg2       time_i[2]:     %d\n",time_i
[2]);
        fprintf(stderr,"dbg2       time_i[3]:     %d\n",time_i
```

```
[3]);
        fprintf(stderr,"dbg2       time_i[4]:     %d\n",time_i
[4]);
        fprintf(stderr,"dbg2       time_i[5]:     %d\n",time_i
[5]);
        fprintf(stderr,"dbg2       time_i[6]:     %d\n",time_i
[6]);
        fprintf(stderr,"dbg2       time_d:        %f\n",*time_
d);
        }
    if (verbose >= 2 && (*kind == MB_DATA_DATA || *kind == MB_
DATA_NAV))
        {
        fprintf(stderr,"dbg2       longitude:     %f\n",*navlo
n);
        fprintf(stderr,"dbg2       latitude:      %f\n",*navla
t);
        fprintf(stderr,"dbg2       speed:         %f\n",*speed
);
        fprintf(stderr,"dbg2       heading:       %f\n",*headi
ng);
        }
    if (verbose >= 2 && *error <= MB_ERROR_NO_ERROR
        && *kind == MB_DATA_DATA)
        {
        fprintf(stderr,"dbg2       nbath:      %d\n",
            *nbath);
        for (i=0;i<*nbath;i++)
          fprintf(stderr,"dbg2       beam:%d  flag:%3d  bath:%
f  acrosstrack:%f  alongtrack:%f\n",
            i,beamflag[i],bath[i],
            bathacrosstrack[i],bathalongtrack[i]);
        fprintf(stderr,"dbg2        namp:     %d\n",
            *namp);
        for (i=0;i<*namp;i++)
          fprintf(stderr,"dbg2       beam:%d   amp:%f  acrosst
rack:%f  alongtrack:%f\n",
            i,amp[i],bathacrosstrack[i],bathalongtrack[i]);
        fprintf(stderr,"dbg2        nss:      %d\n",
            *nss);
```

```
        for (i=0;i<*nss;i++)
          fprintf(stderr,"dbg2        pixel:%d   ss:%f  across
track:%f  alongtrack:%f\n",
            i,ss[i],ssacrosstrack[i],ssalongtrack[i]);
        }
    if (verbose >= 2)
        {
        fprintf(stderr,"dbg2       error:      %d\n",*error);
        fprintf(stderr,"dbg2  Return status:\n");
        fprintf(stderr,"dbg2       status:     %d\n",status);
        }

    /* return status */
    return(status);
}
```

The other data access functions in mbsys_wassp.c are structured similarly to *mbsys_wassp_extract()*.

Step 6. Integrate the New I/O module Into MBIO

Once the I/O module source files have been written, there are a few modifications to the files:

- mbsystem/src/mbio/mb_format.h
- mbsystem/src/mbio/mb_format.c

that are required to integrate the new I/O module with the MBIO library. The code fragments below include the changes.

The new data system MB_SYS_WASSP must be added to the list of data systems in mb_format.h:

```
/* Supported swath sonar systems */
#define MB_SYS_NONE     0
#define MB_SYS_SB       1
------------lines deleted---------------
#define MB_SYS_3DATDEPTHLIDAR   35
#define MB_SYS_WASSP            36
```

The number of supported formats must be incremented (in this case from 73 to 74) and the new format MBF_WASSPENL must be added to the list of formats in mb_format.h:

```
/* Number of supported MBIO data formats */
#define MB_FORMATS  74

/* Data formats supported by MBIO */
#define MBF_DATALIST    -1
#define MBF_NONE    0
#define MBF_SBSIOMRG    11  /* SeaBeam, 16 beam, bathymetry,
                               binary, uncentered, SI
O. */
#define MBF_SBSIOCEN    12  /* SeaBeam, 19 beam, bathymetry,
                               binary, centered, SIO.
 */
-------------------------lines deleted-------------------------
----------------------
#define MBF_3DDEPTHP    231     /* 3DatDepth processed format
for 3DatDepth LIDAR,
                               variable beams, bathym
etry, amplitude,
                               binary, single files,
3DatDepth. */
#define MBF_WASSPENL    241     /* WASSP Multibeam Vendor Form
at,
                               WASSP multibeams,
                               bathymetry and amplitu
de,
                               122 or 244 beams, bina
ry, Electronic Navigation Ltd. */
```

Prototypes for the MBIO registration functions *mbr_register_wasspenl()* and *mbr_info_wasspenl()* must be added to mb_format.h:

```
/* format registration function prototypes */
int mbr_register_sbsiomrg(int verbose, void *mbio_ptr, int *er
ror);
int mbr_register_sbsiocen(int verbose, void *mbio_ptr, int *er
ror);

----------------lines deleted-----------------------------------
-------

int mbr_register_3ddepthp(int verbose, void *mbio_ptr, int *er
ror);
int mbr_register_wasspenl(int verbose, void *mbio_ptr, int *er
ror);
int mbr_info_sbsiomrg(int verbose,
                      int *system,
                      int *beams_bath_max,
                      int *beams_amp_max,
                      int *pixels_ss_max,
                      char *format_name,
                      char *system_name,
                      char *format_description,
                      int *numfile,
                      int *filetype,
                      int *variable_beams,
                      int *traveltime,
                      int *beam_flagging,
                      int *nav_source,
                      int *heading_source,
                      int *vru_source,
                      int *svp_source,
                      double *beamwidth_xtrack,
                      double *beamwidth_ltrack,
                      int *error);

----------------lines deleted-----------------------------------
-------
```

```
int mbr_info_wasspenl(int verbose,
                        int *system,
                        int *beams_bath_max,
                        int *beams_amp_max,
                        int *pixels_ss_max,
                        char *format_name,
                        char *system_name,
                        char *format_description,
                        int *numfile,
                        int *filetype,
                        int *variable_beams,
                        int *traveltime,
                        int *beam_flagging,
                        int *nav_source,
                        int *heading_source,
                        int *vru_source,
                        int *svp_source,
                        double *beamwidth_xtrack,
                        double *beamwidth_ltrack,
                        int *error);
```

References to the new format must be added to function *mb_format_register()* in mb_format.c:

```
int mb_format_register(int verbose,
            int *format,
            void *mbio_ptr,
            int *error)
{
    char    *function_name = "mb_format_register";
    int status;
    struct mb_io_struct *mb_io_ptr;
    int i;

    ----------------lines deleted-----------------------------
-----

    /* look for a corresponding format */
```

```
    if (*format == MBF_SBSIOMRG)
        {
        status = mbr_register_sbsiomrg(verbose, mbio_ptr, erro
r);
        }

    ----------------lines deleted-----------------------------
-----

    else if (*format == MBF_3DDEPTHP)
        {
        status = mbr_register_3ddepthp(verbose, mbio_ptr, erro
r);
        }
    else if (*format == MBF_WASSPENL)
        {
        status = mbr_register_wasspenl(verbose, mbio_ptr, erro
r);
        }
    else
        {
        status = MB_FAILURE;
        *error = MB_ERROR_BAD_FORMAT;
        }

    ----------------lines deleted-----------------------------
-----

    /* return status */
    return(status);
}
```

References to the new format must be added to function *mb_format_info()* in mb_format.c:

```
int mb_format_info(int verbose,
            int *format,
            int *system,
            int *beams_bath_max,
```

```
            int *beams_amp_max,
            int *pixels_ss_max,
            char *format_name,
            char *system_name,
            char *format_description,
            int *numfile,
            int *filetype,
            int *variable_beams,
            int *traveltime,
            int *beam_flagging,
            int *nav_source,
            int *heading_source,
            int *vru_source,
            int *svp_source,
            double *beamwidth_xtrack,
            double *beamwidth_ltrack,
            int *error)
{
    char    *function_name = "mb_format_info";
    int status;
    int i;

----------------lines deleted-----------------------------------
-------

    /* look for a corresponding format */
    if (*format == MBF_SBSIOMRG)
        {
        status = mbr_info_sbsiomrg(verbose, system,
            beams_bath_max, beams_amp_max, pixels_ss_max,
            format_name, system_name, format_description,
            numfile, filetype,
            variable_beams, traveltime, beam_flagging,
            nav_source, heading_source, vru_source, svp_source
,
            beamwidth_xtrack, beamwidth_ltrack,
            error);
        }

----------------lines deleted-----------------------------------
```

```
-------

    else if (*format == MBF_3DDEPTHP)
        {
        status = mbr_info_3ddepthp(verbose, system,
            beams_bath_max, beams_amp_max, pixels_ss_max,
            format_name, system_name, format_description,
            numfile, filetype,
            variable_beams, traveltime, beam_flagging,
            nav_source, heading_source, vru_source, svp_source
,
            beamwidth_xtrack, beamwidth_ltrack,
            error);
        }
    else if (*format == MBF_WASSPENL)
        {
        status = mbr_info_wasspenl(verbose, system,
            beams_bath_max, beams_amp_max, pixels_ss_max,
            format_name, system_name, format_description,
            numfile, filetype,
            variable_beams, traveltime, beam_flagging,
            nav_source, heading_source, vru_source, svp_source
,
            beamwidth_xtrack, beamwidth_ltrack,
            error);
        }
    else if (*format == MBF_DATALIST)
        {
        *format = MBF_DATALIST;
        *system = MB_SYS_NONE;
        *beams_bath_max = 0;
        *beams_amp_max = 0;
        *pixels_ss_max = 0;
        strcpy(format_name,"MBF_DATALIST");
        strcpy(system_name,"MB_SYS_DATALIST");
        strcpy(format_description,"MBF_DATALIST");
        strncpy(format_description, "Format name:          MBF
_DATALIST\nInformal Description: Datalist\nAttributes:
   List of swath data files, each filename \n\tfollowed by MB-
System format id.\n", MB_DESCRIPTION_LENGTH);
```

```
        *numfile = 0;
        *filetype = 0;
        *variable_beams = MB_NO;
        *traveltime = MB_NO;
        *beam_flagging = MB_NO;
        *nav_source = MB_DATA_NONE;
        *heading_source = MB_DATA_NONE;
        *vru_source = MB_DATA_NONE;
        *svp_source = MB_DATA_NONE;
        *beamwidth_xtrack = 0.0;
        *beamwidth_ltrack = 0.0;
        status = MB_FAILURE;
        *error = MB_ERROR_BAD_FORMAT;
        }

----------------lines deleted-----------------------------------
-------

    /* return status */
    return(status);
}
```

References to the new format must be added to function *mb_get_format()* in mb_format.c:

```
int mb_get_format(int verbose, char *filename, char *fileroot,
            int *format, int *error)
{
    char    *function_name = "mb_get_format";
    int status = MB_SUCCESS;

----------------lines deleted-----------------------------------
-------


    /* first look for MB suffix convention */
    if (found == MB_NO)
        {
        if (strlen(filename) > 6)
```

```
        i = strlen(filename) - 6;
        else
        i = 0;
        if ((suffix = strstr(&filename[i],".mb")) != NULL
            || (suffix = strstr(&filename[i],".MB")) != NULL)
            {
            suffix_len = strlen(suffix);
            if (suffix_len >= 4 && suffix_len <= 6)
                {
                if (fileroot != NULL)
                {
                strncpy(fileroot, filename, strlen(filename)-s
uffix_len);
                fileroot[strlen(filename)-suffix_len] = '\0';
                }
                if (sscanf(suffix, ".mb%d", format) > 0
                    || sscanf(suffix, ".MB%d", format) > 0)
                found = MB_YES;
                }
            }
        }

----------------lines deleted-----------------------------------
-------

    /* look for a 3DatDepth *.raa file format convention*/
    if (found == MB_NO)
        {
        if (strlen(filename) >= 5)
        i = strlen(filename) - 4;
        else
        i = 0;
        if ((suffix = strstr(&filename[i],".raa")) != NULL)
        suffix_len = 4;
        else if ((suffix = strstr(&filename[i],".RAA")) != NUL
L)
        suffix_len = 4;
        else
        suffix_len = 0;
        if (suffix_len == 4)
```

```
            {
            if (fileroot != NULL)
                {
                strncpy(fileroot, filename, strlen(filename)-s
uffix_len);
                fileroot[strlen(filename)-suffix_len] = '\0';
                }
            *format = MBF_3DDEPTHP;
            found = MB_YES;
            }
        }

    /* look for a WASSP *.000 file format convention*/
    if (found == MB_NO)
        {
        if (strlen(filename) >= 5)
        i = strlen(filename) - 4;
        else
        i = 0;
        if ((suffix = strstr(&filename[i],".000")) != NULL)
        suffix_len = 4;
        else
        suffix_len = 0;
        if (suffix_len == 4)
            {
            if (fileroot != NULL)
                {
                strncpy(fileroot, filename, strlen(filename)-s
uffix_len);
                fileroot[strlen(filename)-suffix_len] = '\0';
                }
            *format = MBF_WASSPENL;
            found = MB_YES;
            }
        }

    /* finally check for parameter file */
    sprintf(parfile, "%s.par", filename);
    if (stat(parfile, &statbuf) == 0)
        {
```

```
        if ((checkfp = fopen(parfile,"r")) != NULL)
            {
            while ((result = fgets(buffer,MBP_FILENAMESIZE,che
ckfp)) == buffer)
                {
                if (buffer[0] != '#')
                    {
                    if (strlen(buffer) > 0)
                        {
                        if (buffer[strlen(buffer)-1] == '\n')
                            buffer[strlen(buffer)-1] = '\0';
                        }

                    if (strncmp(buffer, "FORMAT", 6) == 0)
                        {
                        sscanf(buffer, "%s %d", dummy, &pforma
t);
                        if (pformat != 0)
                            {
                            *format = pformat;
                            if (found == MB_NO)
                                {
                                strcpy(fileroot, filename);
                                found = MB_YES;
                                }
                            }
                        }
                    }
                }
            fclose(checkfp);
            }
        }

----------------lines deleted-----------------------------------
-------

    /* return status */
    return(status);
}
```

Step 7: Update the MB-System Build

In order to update the MB-System build system, we must first add the three new source files to the Makefile.am file in mbsystem/src/mbio/, and then rerun a series of commands from the GNU autotools package to regenerate a number of required files.

With the mbsys_wassp.h, mbsys_wassp.c, and mbr_wasspenl.c files added, the Makefile.am files looks like:

```
lib_LTLIBRARIES = libmbio.la

include_HEADERS = mb_config.h \
    mb_format.h mb_status.h \
    mb_io.h mb_swap.h \
    mb_define.h mb_process.h \

    ---------lines deleted--------------------

    mbsys_benthos.h mbsys_swathplus.h \
    mbsys_3datdepthlidar.h mbsys_wassp.h \
    mbf_sbsiomrg.h mbf_sbsiocen.h \

    ---------lines deleted--------------------

    mbf_mbarirov.h mbf_mbarrov2.h \
    mbf_mbpronav.h mbf_xtfr8101.h

    ---------lines deleted--------------------

libmbio_la_SOURCES = \
        mb_format.c mb_error.c \

    ---------lines deleted--------------------

        mbsys_benthos.c mbsys_swathplus.c \
        mbsys_3datdepthlidar.c mbsys_wassp.c \
        mbr_sbsiomrg.c mbr_sbsiocen.c \

    ---------lines deleted--------------------

        mbr_swplssxp.c mbr_3ddepthp.c \
        mbr_wasspenl.c

    ---------lines deleted--------------------
```

To reconstruct the build system, in a shell execute the following from mbsystem/ (i.e. from the top of the MB-System source tree):

```
#
# Full autotools command sequence after modifying the build sy
stem
#
# First clean up old installation and build
make uninstall
make clean

# Reconstruct the build system
libtoolize --force --copy
aclocal
autoheader
automake --add-missing --include-deps
autoconf
autoupdate

autoreconf --force --install --warnings=all
```

Following these commands, run configure, make, and make install in the usual fashion to build and install MB-System with the new I/O module.

Step 8: Test the New I/O Module

In order to test the functioning of the new I/O module, we attempt to process and display the data sample from UNH/CCOM. The following commands will suffice:

```
#################################################################
# Process the WASSP data
#################################################################
#
# Get datalist  plus ancilliary files
/bin/ls -1 *.000 | awk '{print $1" 241"}' > datalist.mb-1
mbdatalist -o -v
mbdatalist -z

# Get tide models and set for use by mbprocess
mbotps -I datalist.mb-1 -M -D60.0 -V

# set roll and pitch bias
mbset -PROLLBIAS:0.0 -PPITCHBIAS:0.0

# Edit bathymetry
mbedit
mbeditviz -I datalist.mb-1

# Calculate amplitude correction
mbbackangle -I datalist.mb-1 \
        -A1 -Q -V -N87/86.0 -R50 -G2/85/1500.0/85/100
mbset -PAMPCORRFILE:datalist.mb-1_tot.aga

# Process the data
mbprocess
```

Screen grabs of the bathymetry editing are shown here:

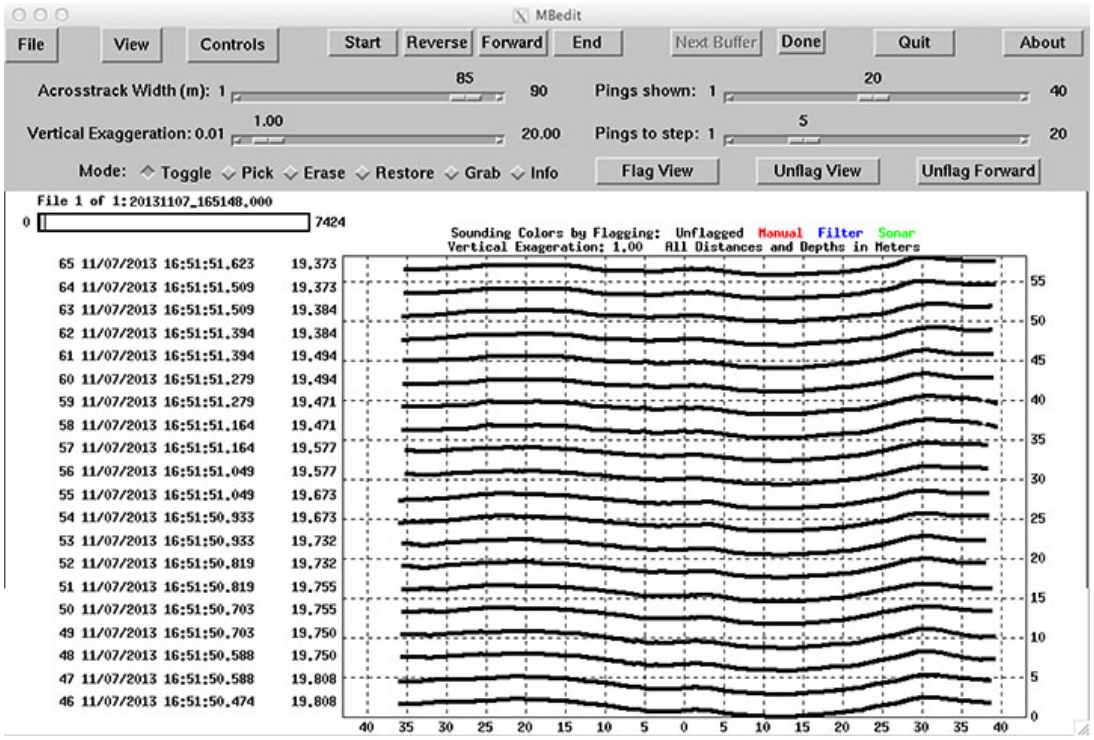


Figure 6. WASSP Multibeam data loaded into mbedit as format 24 1.

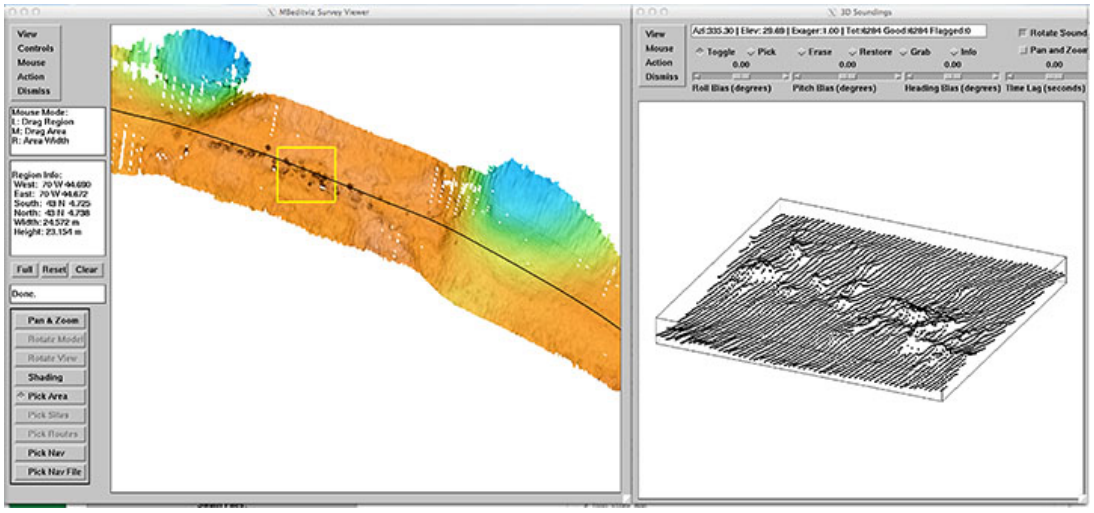


Figure 7. WASSP Multibeam data loaded into mbeditviz as format 241.

Below are some examples of WASSP topography and amplitude maps generated with MB-System through the new I/O module. The associated commands are shown as well.

To generate a topography grid, use mbgrid:

```
# Generate topography grid
mbgrid -I datalistp.mb-1 -A2 -C5 -F5 -N -O ZTopo -V
mbgrdviz -I ZTopo.grd &
```

Below are example maps generated from the topography grid. To generate a bathymetry map with slope shading use mbm_grdplot with the -G5 option:

```
# Topo slope map
mbm_grdplot -I ZTopo.grd \
        -O ZTopoSlope \
        -G5 -D0/1 -A2 \
        -L"WASSP Multibeam Sonar Example":"Topography (meters)
" \
        -MGLfx4/1/43.0/0.5+l"km" \
        -Pa -V
ZTopoSlope.cmd
```

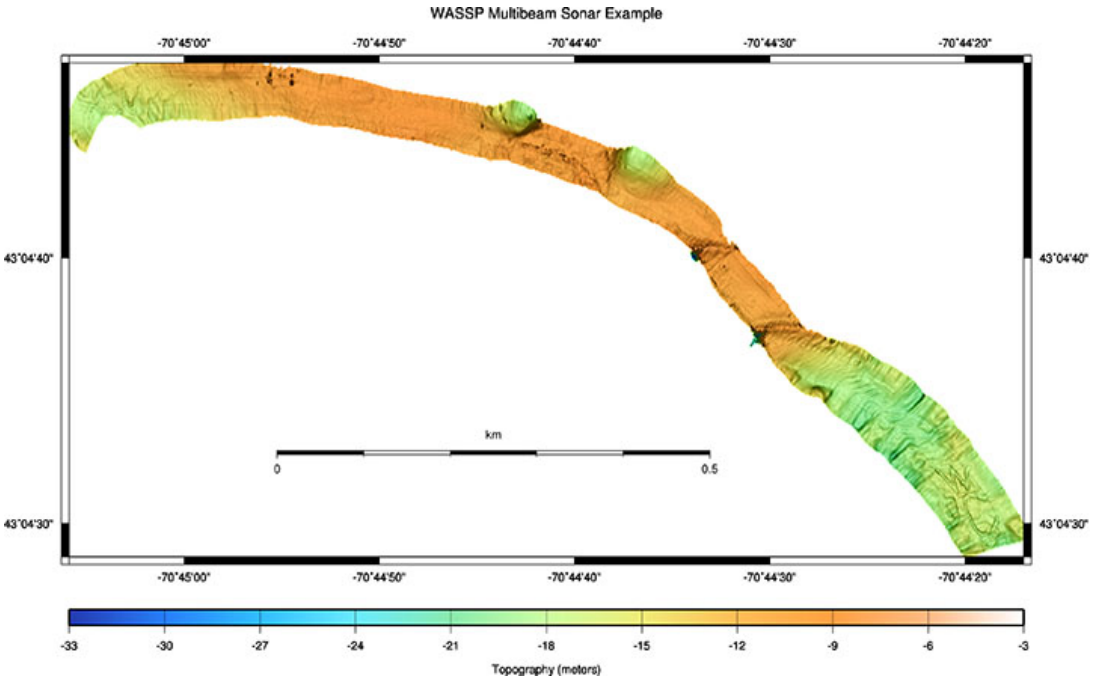


Figure 8. WASSP bathymetry with slope shading

To generate a bathymetry map with contours use mbm_grdplot with the -G1 and -C options:

```
mbm_grdplot -I ZTopo.grd \
        -O ZTopoCont \
        -G1 -C1 -MCW0p -A2 \
        -L"WASSP Multibeam Sonar Example":"Topography (meters)
" \
        -MGLfx4/1/43.0/0.5+l"km" \
        -Pa -V
ZTopoCont.cmd
```

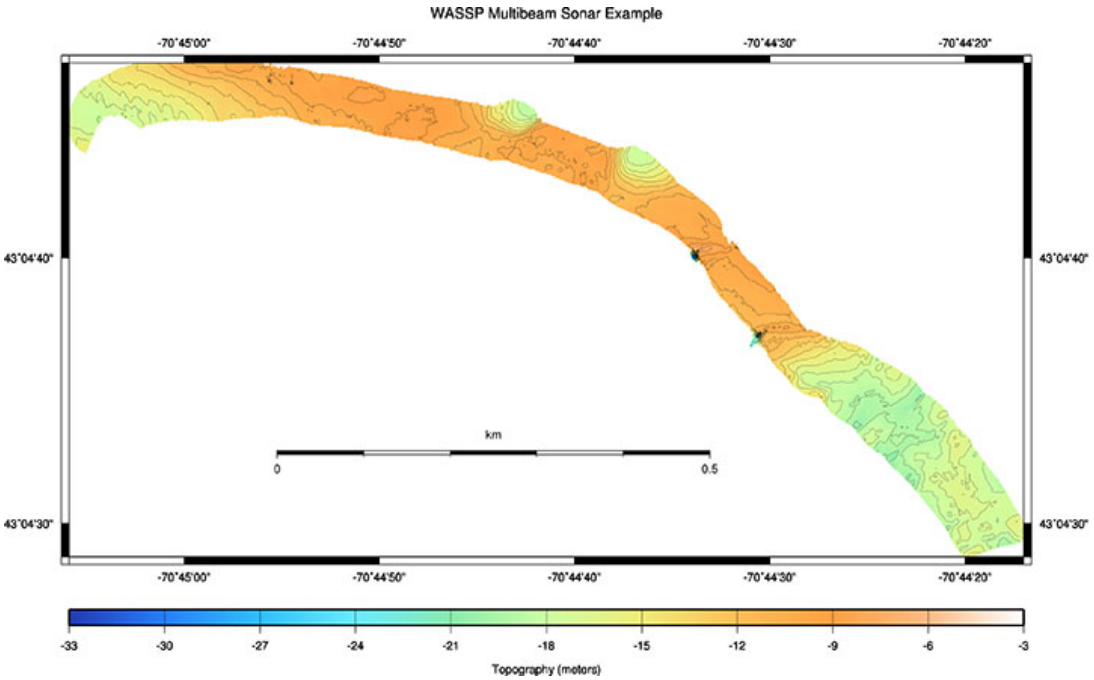


Figure 9. WASSP bathymetry with contours

To add navigation lines to the slope shaded map, add the -MNI option:

```
mbm_grdplot -I ZTopo.grd \
        -O ZTopoSlopeNav \
        -G5 -D0/1 -A2 \
        -L"WASSP Multibeam Sonar Example":"Topography (meters)
" \
        -MGLfx4/1/43.0/0.5+l"km" \
        -MNIdatalistp.mb-1 \
        -Pa -V
ZTopoSlopeNav.cmd
```

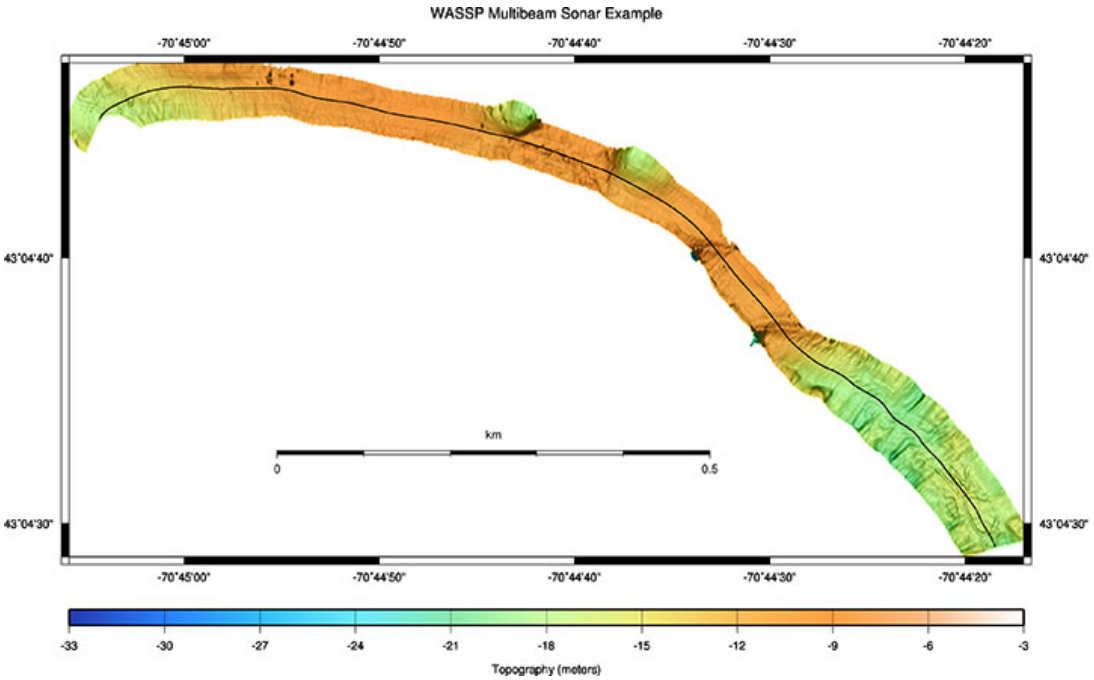


Figure 10. WASSP bathymetry with slope shading and navigation

To generate a corrected amplitude grid, use mbmosaic:

```
# Generate first cut mosaics and maps
mbmosaic -I datalistp.mb-1 -A3 -N -Y2 -F0.05 \
        -O ZAmpC -V
mbgrdviz -I ZTopo.grd -J ZAmpC.grd &
```

To generate a grayscale amplitude map use mbm_grdplot with the -G1 and -W1/4 options:

```
# Generate first cut mosaics and maps
mbmosaic -I datalistp.mb-1 -A3 -E1/0! -N -Y2 -F0.05 \
        -O ZAmpC -V
mbm_grdplot -I ZAmpC.grd \
        -O ZAmpCPlot \
        -G1 -W1/4 -D -S \
        -L"WASSP Multibeam Sonar Example":"Multibeam Amplitude
" \
        -MGLfx4/1/43.0/0.5+l"km" \
        -Pa -V
ZAmpCPlot.cmd
```

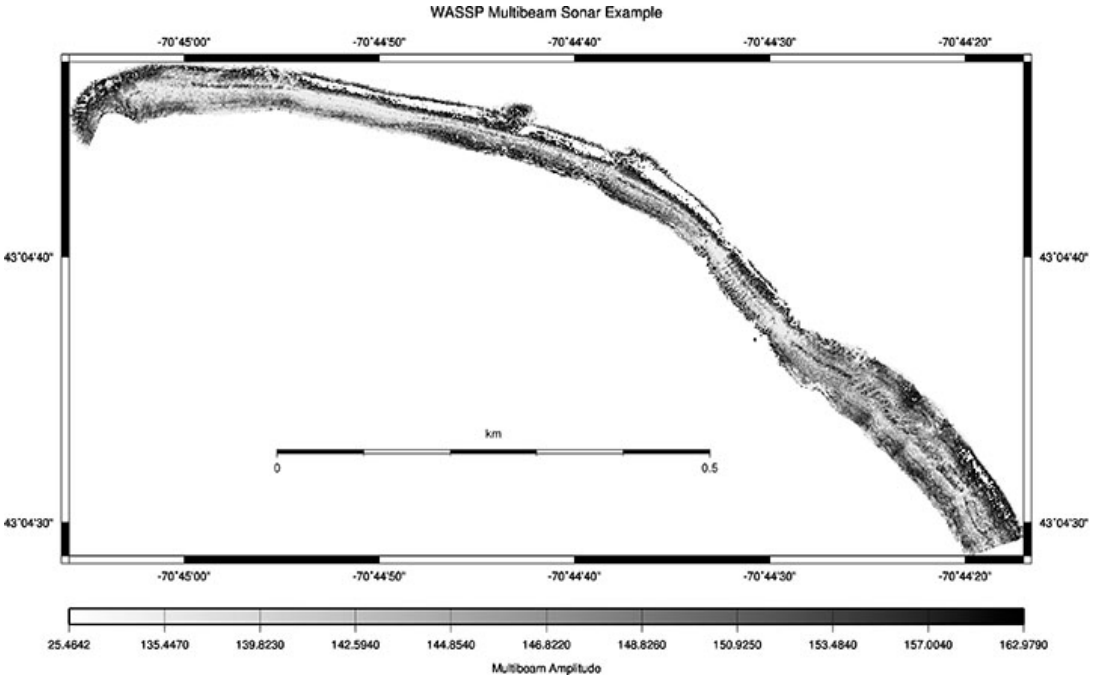


Figure 11. WASSP Corrected Amplitude