sy_io, a package for SEGY I/O

Rolf Maier

ABSTRACT

The *sy_io* package is designed to simplify the creation and reading of SEGY files as much as possible, yet has a sufficient complement of optional variables to allow reading files of any data type or byte order, non-standard trace header length, varying trace data lengths, arbitrary header word locations and types, or missing text and binary headers.

INTRODUCTION

Creating SEGY files is easy, reading them, less so. Incomplete adherence to the standard where this was intended is one reason, but the need to adhere to a company standard that may conform to the days predating any SEGY standard, and certainly predating the appearance of SEG-Y standard revision 1 (Norris and Fainchney, 2002), is a more likely cause of non-standard formats.

The intent of this package is to take from a programmer, and ultimately user, as much tedium as possible when handling SEGY formats, while providing all the flexibility needed with certain data. This is achieved in two ways. The number of functions required to do SEGY I/O is quite small, and so is the number of required parameters. If required, the type, location and value of all header words can be set or reset via an extensive input file which may be broken into pieces as there are separate parsers for three different sections.

For the convenience of a programmer, all but a couple of small, internal functions come with their own man pages. There are also three manuals which describe much of the package in general, and give varying amounts of details. The first of them, and one of the included examples, should suffice to write a program. Sample code is heavily annotated; actual code takes a small portion of the total space.

When reading files, the described package attempts to take from the user as much tedium as possible by interpreting the binary header and reading the number of traces from individual trace headers, unless the trace header promises all traces to have the same size, a flag that can be overwritten. When writing files, the package offers structures for both the file header and trace headers. Values in the structures are set by the user and mapped to the headers with the correct type and offset. The same mechanism works in reverse.

DESCRIPTION

On a given platform, there are two major factors which determine the flexibility of a data transfer package; how easily can it be applied, and how much information is available to the hidden parts of a function that reads or writes a gather, say, to do things right.

The answer to the first part is straight forward. There are functions to read or write either gathers or individual traces. In the case of reading a whole gather, limited sets of samples and traces may be read, and what constitutes a gather can be determined by any header word and a flag indicating whether it should increase, decrease, or be constant.

The question of what information is available takes more effort to answer. All information pertaining to one file is kept in one large structure which may be considered to consist of three parts. Part one contains general variables like $swap$, $nsmp$, max_fold, the maximum number of traces per gather, etc.

Part two describes header words themselves, their location and type. Variables prefixed $TL\_$ for trace header locations, and $TT\_$ for trace header type, respectively, and similar for the file header, all predefined to standard values and resettable with a function $sy\_reset\_hdr\_locs$ to fit real SEGY files. In Segy revision 1 there are only six unassigned bytes, yet there is no assignment for an important value like first break, and there may be others you desire. In this case it is possible to undefine some named header identifiers ($TL\_$ and $TT\_$ parameters) by setting the location, TL_xxx, to zero, and using one of twenty generically named terms like TL_User1. Any header location may contain any type of a number.

The third part is in essence a modified copy of the second part. Since real numbers of any kind are rarely found in headers, file header word FL_xxx is associated with a four-byte integer called FHV_xxx , and similarly for trace header headers: TL_xxx associates with FHV_xxx. Reading and writing trace headers is now easy: to write a header, set non-zero THV_xxx values, then write them with a single call to the function THV_write, and similarly to read a trace header with THV_read.

Trace data are easier to handle because they are all of the same type, specified in part one of the structure. Note that a trace header and data are returned in a single call, a program should not have to do this piecemeal, or the above-mentioned niceties are lost.

In practice, then data I/O works as follows. When writing trace-by-trace, headers may be set as described above, THV_write will map the integers in the structure to a user-supplied array. Repeat the procedure to write a header array for a whole gather. Then sy_write_gather will write a gather, $sy\_writeTrace$ will write a trace. The UNIX man pages come with the package state what these commands do to headers to make a consistent file and, in the first case, a gather.

MANUALS

More than 80 manuals describe just about every facet and little function. Most of these are only required if changes of the underlying code are desired. To use the package read *sy_io_read1st, then perhaps the top sections of* sy_io which has a long list describing in brief far more functions than most users will ever call, and, to get more information on the structure, $sy\_io.h$ (the manual by that name, not the include file).

REQUIRED CODE

For an easy start with no surprises, begin with one of the files listed below. All examples can be compiled with $Makefile.sy$, which should be in the same location as the code examples.

The simplest is $sy\_test.c$. This file, consisting mainly of comments, writes and then reads back a very tiny SEGY file, one gather at the time. $sy\_write\_3d.c$ does I/O trace by trace. Since a program must allocate space for trace headers even though sy_io commands will make the headers, this may save some space when arrays get huge, as in the case of 3D data. $sy\_test\_ft.f90$ is a Fortran version of the writing portion of the previous example. Due to incomplete Fortran support, the I/O is handled via a C subroutine, $eff\_main.c$, while the work is done by $sy\_test\_ft\_worker.f90$.

INSTALLATION

Installation instructions are enclosed with the software. Once installed, anyone wanting to use the package should be aware of, and have access to, the files mentioned in the last section, the library file $libsy\_io.a$ which $Makefile.sy$ refers to, and the include files $sy\_io.h$, $segy\_param.h$, $HdrLocs.h$, $FHV.h$, and $THV.h$, only the first of which a programmer includes.

SUPPORTING COMMANDS

The huge number of available parameters, combined with the fact that many have sensible defaults (header word locations and types), or are used optionally (trace header variables prefixed $THV\_$), enforces a way to order and economize. First, there has been the conceptial grouping of all variables belonging to a file into three groups. Second, the variables are named in the input file, they may be arranged in any order, and any variable with an acceptable default may, of course, be omitted. Third, there are special little commands to help handling the variables.

sy_pr_txthdr prints a template text header.

sy_print_params prints the whole long list of variables, with explanations which may be added to or deleted.

$sy\_hdr\_byte\_coverage$ checks how often any header byte is referred to. These numbers should be zero or one, but not larger.

A large number of manuals are available. Programmers will likely get all they need reading only the first, *sy_io_read1st, of the three manuals listed above in the section on manuals. The second manual,* sy_io, contains an extensive list of functions, together with a short description. Beyond that, each function has its own manual. Reading the sample programs will be sufficient in most cases.

DISCUSSION

The described package of C commands allows SEGY I/O with little programming, yet has sufficient support to handle the regular, non-standard files, regardless of bytes per trace header, endianness, variable trace length, non-standard header words, and any data type not exceeding four bytes.

REFERENCES

Norris, M. W., and Fainchney, A. K., Eds., 2002, SEG Y rev. 1 Data Exchange Format: SEG Technical Standards Committee Release 1.0.