*Stanford Exploration Project, Report 70, December 18, 1997, pages 1–??*

*Stanford Exploration Project, Report 70, December 18, 1997, pages 1–??*

# Introduction to seplib and SEP utility software

*Jon Claerbout*[1]

***keywords:*** *algorithm, graphics, processing*

**ABSTRACT**

Most of the seismic utility software at the Stanford Exploration Project handles seismic data as a rectangular lattice or "cube" of numbers. Each cube processing program appends to the history file for the cube. Preprocessors extend fortran to enable it to allocate memory at run time, to facilitate input and output of data cubes, and to facilitate self-documenting programs.

At Stanford Exploration Project (SEP) a library of subroutines known as "seplib" evolved for routine operations. These mostly handle data in the form of cubes, planes, and vectors. A cube is defined by 14 parameters with standard names and two files: one the data cube itself, and the other containing the 14 parameters and a history of the life of the cube as it passed through a sequence of cube processing programs. These cube processing programs are mostly written by researchers but several nonscientific cube programs have become highly developed and are widely shared. Altogether there is (1) a library of subroutines, (2) a library of main programs, (3) some naming conventions, and (4) a graphics library called `vplot` described by Dellinger [1989]. The subroutine library has good manual pages. The main programs rarely have manual pages, their documentation being supplied by the on-line self documentation that is extracted from the comments at the beginning of the source file. Below is a list of the names of popular main programs.

**Byte** scales floats to brightness bytes for raster display.

**Cat** concatenates conforming cubes along the 3-axis.

**Contour** contour plot a plane.

**Cp** copy a cube.

**Dd** convert between floats, complex, bytes, etc.

**Dots** plots a plane of floats.

[1]**email:** not available

**Ft3d** three-dimensional Fourier transform.

**Graph** plots a line of floats.

**In** checks the validity of a data cube.

**Merge** merges conforming cubes side-by-side on any axis.

**Movie** is Rick's cube viewer. (Needs evaluation under X).

**Noise** adds noise to data.

**Reverse** reverses a cube axis.

**Spike** makes a plane wave of synthetic data.

**Ta2vplot** converts byte format to attractive section vplot.

**Tpow** scales data by a power of time $t$ (1-axis).

**Thplot** hidden line plot.

**Transpose** transposes axes.

**Tube** view vplot plot on a screen.

**Wiggle** plots a plane of floats.

**Window** finds a subcube by truncation or subsampling.

To use cube processing programs, read this document, and then for each command, read its on-line self documentation. To write cube processing programs, read the manual page for `seplib` and the subroutines mentioned there and here. To write `vplot` programs, see Dellinger's [1989] documentation.

## THE DATA CUBE

The data cube itself is like a Fortran three-dimensional matrix. Its location in the computer file system is denoted by "`in=PATHNAME`" where "`in=`" is the literal occurrance of those three letters and "`PATHNAME`" is a directory tree location like "/sep/professor/pvi/data/western73.F". Like the Fortran cube, the cube can be real, complex, double precision, or byte, and these cases are distinguished by the element size in bytes. Thus the history file contains one of "`esize=4`", "`esize=8`", or "`esize=1`". Embedded blanks around the "=" are always forbidden. The cube values are binary information; they cannot be printed or edited (without the intervention of something like a fortran "format"). To read and write cubes, see the manual pages for such routines as `reed, sreed, rite, srite, snap`.

A cube has three axes. The number of points on the 1-axis is `n1`. A fortran declaration of a cube can be `real mydata(n1,n2,n3)`. For a plane, `n3=1`, and for a line, `n2=1`, and many programs take "1" as the default for an undefined value of `n2` or `n3`. The physical location of the single data value `mydata(1,1,1)`, like a mathematical origin $(o_1, o_2, o_3)$, is denoted by the three real variables `o1`, `o2`, and `o3`. The data cube values are presumed to be uniformly spaced along these axes like the mathematical increments $(\Delta_1, \Delta_2, \Delta_3)$ which may be negative and are denoted by the three real variables `d1`, `d2`, and `d3`.

Each axis has a label and naturally these labels are called `label1`, `label2`, and `label3`. Examples of labels are `kilometers`, `sec`, `Hz`, and `"offset, km"`. Most often, `label1="time, sec"`. Altogether that is $2 + 3 \times 4$ parameters, and there is an optional title parameter that is interpreted by most of the plot programs. An example is '`title="Yilmaz and Cumro Canada profile 25"`'. We reserve the names `n4,o4,d4,label4`, (a few programs support them already) and please don't use `n5` etc for anything but a 5-dimensional cubic lattice.

## THE HISTORY FILE

The 15 parameters above, and many more parameters defined by authors of cube processing programs lie on the history file (which is ASCI so you can print it). A great many cube processing programs are simple filters, i.e. one cube goes in and one cube comes out and that is the case I'll describe in detail here. For other cases such as where two go in and one comes out, or none go in and one comes out (synthetic data) or one goes in and none come out (plotting program), I refer you to the manual pages, particularly subroutine names begining with `aux` (as in auxiliary).

Let us dissect an example of a simple cube processing program and its use. Suppose we have a seismogram in a data cube and we want only the first 500 points on it, i.e. the first 500 points on the 1-axis. A utility cube filter named `Window` will do the job. Your command line looks like:

```
< mygiven.H Window n1=500 > myshort.H
```

On this command line, `mygiven.H` is the name of the history file of the data you are given, and `myshort.H` is the history file you will create. The moment the Window program, or any other `seplib` program, begins, it copies `mygiven.H` to `myshort.H` and from then on, information can only be appended to `myshort.H`. When `Window` learns that you want the the 1-axis on your output cube to be 500, it does "`call putch('n1','i',500)`" which appends `n1=500` to `myshort.H`. But before this, some other things happen. First, `seplib`'s internals will get your name, the date, the name of the computer you are using, and `Window`'s name (which is `Window`), and append those to `myshort.H`. Then, `seplib`'s internals will scan `mygiven.H` for `in=somewhere` to find the input data cube itself, and then it will figure out where you want to keep the output cube. It will guess that someone named professor wants to keep his data cube at some place like `/scr/professor/_Window.H@`. You should read the manual

page for `datapath` to see how you can set up the default location for your data sets. A lot of experience and skill went into designing `datapath` to save you from many aggravating real-world difficulties.

When a cube processing filter wonders what is the value of `n1`, for the cube coming in, it makes a subroutine call like "`call hetch("n1","i",n1)`". The value returned for `n1` will be the *last* value of `n1` found on the history file. The window program also needs to find a different `n1`, the one you put on the command line. For this it will invoke something like "`call getch("n1","i",n1out)`". Then, so the next user will know how big the output cube is, it will "`call putch("n1","i",n1out)`". For more details, see the manual pages.

If you want to take input parameters from a file instead of from the command line, you can use something like, "`<in.H Window par=myparfile.p > out.H`". The "`.p`" is my naming convention and it is wholly optional as is the "`.H`" notation for a History file.

Sepcube programs are self-documenting. When you type the name of the program with no input cube and no command line arguments, you should see the self-documentation (which comes from the initial comment lines in the program).

Finally, I'll mention pipelineing. We can slice a plane out of a data cube, make a contour plot, and display the plot, all on one command line with this:

```
<in.H Window n3=1 | Contour | Tube
```

where like UNIX pipes, the "`|`" denotes the passage of information from one program to the next. The history files do flow down UNIX pipes. You may not have noticed that someplace had to be assigned to the data at the intermediate stages, and when you typed the line above you were spared that clutter. To write `seplib` programs that allow pipelining, you need to read the manual page on `hclose()` to keep the history file from intermingling with the data cube itself.

A sample history file follows: This was an old one so I removed a few anachronisms manually.

```
# Texaco Subduction Trench: read from tape by Bill Harlan
n1=1900 n2=2274
o1=2.4 it0=600 d1=.004 d2=50.  in=/d5/alaska

Window:   bill   Wed Apr 13 14:27:57 1983
input() :    in ="/d5/alaska"
output() : sets next in="/q2/data/Dalw"
Input:  float Fortran (1900,2274,1)
Output: float Fortran (512,128,1)
 n1=512 n2=128 n3=1

Swab:   root@mazama   Mon Feb 17 03:23:08 1986
# input history file /r3/q2/data/Halw
input() :    in ="/q2/data/Dalw"
```

```
output() : sets next in="/q2/data/Dalw_002870_Rcp"
#ibs=8192 #obs=8192

Rcp: paul Mon Feb 17 03:23:15 PST 1986
Copying from mazama:/r3/q2/data/Halw
to hanauma:/q2/data/Halw
in="/q2/data/Dalw"

Cp:   jon@hanauma   Wed Apr  3 23:18:13 1991
input() :    in ="/q2/data/Dalw"
output() : sets next in="/scr/jon/_junk.H@"
```

## MEMORY ALLOCATION

Sepcube programs can be written in fortran or C. A serious problem with fortran is that memory cannot be allocated for arrays whose size is determined at run time. We have gotten around this limitation by using two home-grown preprocessors, one called saw (Stanford Auto Writer) for main programs, and one called sat (Stanford Auto Temporaries) for subroutines. The sat preprocessor allows you to declare temporary arrays of arbitrary dimension such as

```
temporary real*4 data(n1,n2,n3), convolution(j+k-1)
```

The saw preprocessor also calls an essential initialization routine initpar(), organizes the self-doc, and simplifies data cube input. Both preprocessors transform either fortran or ratfor. See the on-line self documentation or the manual pages for more details.

## ACKNOWLEDGMENTS

Rob Clayton wrote the original parameter fetching program named getpar. After I split his getpar into getch and hetch, Stew Levin finished the library properly, doing far more than I can recount. The cube main program utilities were written and debugged by many people, spanning over a decade. The manual pages were mostly written by Stew Levin, Rick Ottolini, and for vplot, Joe Dellinger. Rob Clayton and Dave Hale designed vplot, but Joe Dellinger gave it sustained life by setting it in a device independent form and incorporating many important features. I prototyped saw and sat and Biondo Biondi completed them. Dave Nichols contributed to sat and provided vital maintainence functions and introduced cake. In 1991, Benoit du Boullay at the French Petroleum Institute took a copy of the entire library with the goal of merging in some of the work done in France. We expect to see that new version back at Stanford by mid 1991 and plan to continue growing from there.

You haven't read anything here about science, nothing but management---just the kind of trivia that often prevents outstanding theoreticians from doing outstanding applied work. So I acknowledge Rick Ottolini. My envy of his ability to assemble prototype code of a new idea before the rest of us finished talking about it led me to begin this all from his ``ricklib''.

## REFERENCES

Dellinger, J., 1989, Why does SEP still use vplot? SEP--**61**, p. 327--335.