REVISION RECORD			
Rev	Description	Date	Author
A	Initial Release	13-Nov-03	E. Wolfe

Contents

1. References 3

2. Introduction 3

2.1 Intended Audience 3

2.2 Document Conventions 3

3. Program Structure 3

3.1 Overview 3

3.2 Pseudo-Code 5

3.3 Files 6

4. Theory of operation 8

5. Maintenance Activities 9

5.1 Extending 9

5.2 Compiling and Building 9

Appendix A, Auxiliary Programs 9

Tables

Table 1: Subroutines and Functions Required 6

1. References

1) Gell, D. “Profile File Format”, SPRL File 055-3532.

2) Gell, D. “Vector File Format”, SPRL File 055-3933.

3) Gell, D. “Vector Requirements”, SPRL File 055-4020.

4) Skinner, W.R., “Revised VECTOR algorithm”, SPRL File 055-4263.

5) Gell, D. ,“File Naming Convention”, SPRL File 055-3545.

6) Gell, D. ,“Track Angle Computation”, SPRL File 055-4114.

7) Wolfe, E., “FW Configs”, SPRL File 055-4171.

8) Gell, D., “Scan File Format”, SPRL File 055-3527.

9) Wolfe, E., “NetCDF Lore”, SPRL File 055-4058.

10) Wolfe, E., “Vector User’s Guide”, SPRL File 055-4279.

11) Wolfe, E., “Utility Routines”, SPRL File 055-4057.

2. Introduction

The purpose of this document is to educate the maintenance programmer about the vector program so that s/he can:

- Correct any errors that are found.
- Modify the behavior of the program.
- Rebuild the program as needed when support modules (system libraries, TIDI libraries and object files, etc.) are modified.

2.1 Intended Audience

This document assumes that the reader is a programmer with a good working knowledge of the Fortran programming language, reasonable facility with the Unix operating system, and some understanding of the TIDI data system. An understanding of the netCDF file format is also needed.

2.2 Document Conventions

3. Program Structure

3.1 Overview

Basically, vector collects (read_prof()) inverted profile data into variables with shapes expected by the routines define_reg_Trackmap(), map_scalar_values(), and determine_vector(). These line

of sight winds are then converted to meridional and zonal winds by determine_vector(reference 4)) which interpolates them onto a regularly spaced track angle grid.

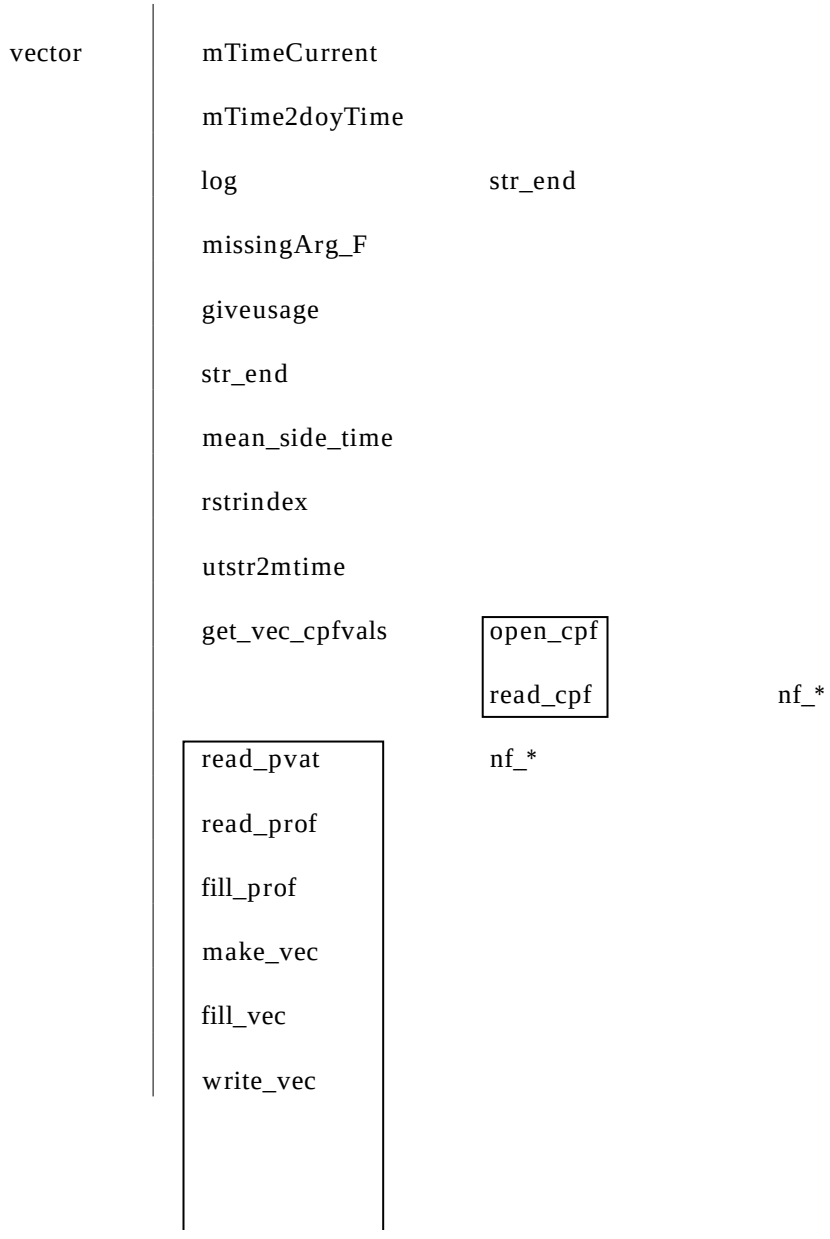
Ancillary data (time, lat, lon, etc) are interpolated onto the grid map.

Then, one altitude at a time, the los speed is separated into its components by determine_vector(), and the scalar quantities temperatures, emission rates, and backgrounds are processed by map_scalar_values().

The processed values are then copied into an array of vector structures which is then written (write_vec()) to a netCDF formatted file.

The subroutines and functions used by vector, and their locations, are listed in Table 1: Subroutines and Functions Required. Reference (11) contains more information.

3.2 Calling Tree



chek_cdftypes		
set_tidi_atts		
reg_trackMap		
polint		
determine_vector	axb	lubksb_a
		ludcmp_a
map_scalar_values	add_var	

3.3 Pseudo-Code

```
 Initialization (any error logs message, exits with status = Error)

Collect command line arguments
Open input Profile file, create output Vector file
Get CPF values and PVAT values

                              Processing
While there are unread input records

     collect input record indices and tel_ids until
     the scan table id or the flight direction changes, or the
     mission time exceeds the last mission time read by 10 minutes

     for side = cold_side, warm_side

       make a subset of the collected profile file indices based on
       tel_ids for this side

       read the subset of profiles into array of profile structures

       generate an evenly spaced grid of track angles for this
       subset

       interpolate ancillary data onto the map and store in vector
       records

       for each altitude

         decompose the speed into its u and v components at each
         measurement
         copy mapped u and v values to output structures
            (bad values are indicated by large variances)


         if present, process the "scalar" quantities:
         temperature, emission rate, background

           if status from map_scalar_values is good
             copy values to output structures
           else
             leave output values to previously set missing values
             increment the p_status flag
```

```
            endif
          endif

        endfor each altitude

        write this subset of vector records to the output file

      endfor cold or warm

end while have unread records

                                  Clean up:
Close input and output files
Write processing statistics
Exit with appropriate error code
```

3.4 Input / Output Files

.PRF input file containing profiles to be placed on a regularly spaced track angle grid.

.pvat daily file containing "as flown" orbit information, for inclination and right ascension node.

.cpf constant parameter file containing values for PI, Earth radius, etc.

.VEC profiles mapped to regularly spaced track angle grid. See reference (2) for full list of products.

Table 1: Subroutines and Functions Required

Subroutine	*Purpose*	*Location*
time calculation		
mTimeCurrent	calculate current time as mission time	missionTime.a
mTime2doyTime	convert mission time to day-of-year format	missionTime.a
mean_sid_time	compute mean sidereal time	vector/
messaging and command line parsing support		
log	issue standard format error messages	util.a
giveusage	brief message on how to use this program	vector.F
str_end	finds end of a string	util.a
rstrindex	finds pattern in string, searching from last character to first	util.a
missingArg_F	check for lack of argument when	vector.F

	command line switch needs one	
utstr2mtime	convert hh:mm:ss to total seconds for start/end time command line option	vector/
ancillary data		
get_vec_cpfvals	read some constants	vector/
read_pvat	read orbit data	vector/
netCDF file routines		
read_prof	read a profile data file	read_prof/
fill_prof	read netCDF file to fill a profile structure with missing values	vector/
make_vec	create netCDF file, define variables	vector/
write_vec	write one tidi vector structure	vector/
fill_vec	read netCDF file to fill a tidi_vector structure with missing values	vector/
chek_cdftypes	test if netCDF variable type is same as certain attributes and if attribute is defined	util.a
set_tidi_atts	set some global netCDF attributes	util.a
nf_*	various netCDF routines	libnetcdf.a
calculate vector products		
reg_trackMap	generate an evenly-spaced track angle map	vector/
polint	polynomial interpolation	vector/
determine_vector	calculate meridional (v) & zonal (u) winds from line of sight measurements	vector/
map_scalar_values	map a scalar quantity on a specified grid	vector/
matrix calculation		
axb	solve the linear equation AX = B	vector/
ludcmp_a	do LU decomposition of matrix A	
lubksb_a	solve AX = B, with A decomposed	
add_var	add variance	vector/

4. Theory of operation

Vector is a fairly simple and straightforward program, except for the bookkeeping details.

The first thing that is done is collect input arguments. An input file name is required. Other options are permitted, including “help” to show what these options are, and “tell” to just show filenames and paths then exit. There are also some debugging options. Unless full paths are specified for the input or output filenames, the full paths are generated according to TIDI project specifications (build_tidi_filename()).

The input file is opened, the output file is created, and the CPF and PVAT files are read.

A vector record is filled with “missing_values” (fill_vec()) and maintained to reinitialize the working array of records for each new subset of profile records that are processed. If there are no records in the input file, this record is written to the output file, and the program exits.

Processing is done inside of a large while loop that continues until the input file is exhausted. The processing routines use working arrays shaped as an array of maxpoints orws by two(telescopes): array(maxpoints, twotelescopes), but the input data contains four telescopes. These are treated in pairs, according to which side of the instrument they occupy. This is accomplished by collecting indices and copying the input records into the working arrays.

The current input record is read to collect time, table_id, and flight_dir. A short while loop is entered to record the input file record index, telescope_id, and increment the counter of the number of records collected for this scan table (n_scnrecs); then increment the input record index and collect the next time, table_id, and flight_dir. This loop continues until the newly-read table_id or flight_dir changes, or the next time is more than 10 minutes later than the last time read.

Another while loop is entered to process the cold side (telescopes 1 & 2) then the warm side (3 & 4) data. For each of the n_scnrecs the telescope_id (45, 135, 225, 315) is converted to an index: 1 or 2. If the telescope is on the current side: its index is recorded in the telnums array to serve as indices into the second dimension of the data_vals array, the n_sameside counter is incremented and the input file record number is recorded in the samesiderec array, and the counter for this telescope is incremented and recorded in the pointnums array to serve as the index into the first dimension of the data_vals array. Another array, rev_idx, records the current value of the n_sameside counter and serves to map the indices held in pointnum and telnum back into the array of input file record numbers to facilitate processing the in_saa and data_ok flags.

The input records specified for this side by samesiderec are then read into an array of profile records (structures). The trackangles from this are copied to working arrays and passed to reg_trackMap() to generate an evenly spaced grid enclosing the trackangles.

Ancillary data (time, lat, lon, etc) are then copied to working arrays and interpolated (polint()) onto this map.

A loop is entered to process data one altitude at a time to convert speed to meridional and zonal winds and interpolate the values onto the map (determine_vector()). The scalar quantities, temperatures, backgrounds, volume emission rates are interpolated onto the grid.

When the altitude loop ends, the mapped quantities are written to the output file and control passes back to the loop for the two sides. When that loop ends, control passes back to the main loop and another set of records is collected.

5. Maintenance Activities

5.1 Extending

Should products be added to the profile files that need processing by vector, it should be fairly straightforward to add them. Each variable is copied to arrays dimensioned according to the expectations of the processing subroutines, processed, then copied to the array of vector structures. Each variable has a comment indicating the start of that section, so it should be a simple matter of copy/pasting the code and changing the input and output names to the new product.

Those variables not dimensioned with altitude, such as time, lza, ilat and, ilon, are processed before the loop for each altitude begins, and are usually interpolated by polint(). Lat and lon are calculated from the track angle map values. The text flags cannot be interpolated and are handled individually.

Variables dimensioned with altitude are processed within the altitude loop with a call to map_scalar_values – except for speed.

5.2 Compiling and Building

This program is compiled using the f77 Fortran compiler. It is crucial to note that the module file names end with < .F > , a capital letter, in order to invoke the preprocessor to properly process include file directives and other directives needed to accommodate more than one machine architecture.

Copies of the modules needing change should be made in the programmer's sandbox by checking them out using the mkssi program on the Hpux machines. Once changes are made, the simple command "make" is issued and any changed modules are recompiled and all required modules are linked to form the executable. If include files were added or removed from any modules, the command "make depend" should be issued first to update the list of dependencies on the include files (only needs to be done once, the platform does not matter). The makefile will determine which machine architecture is the target (HPUX or Sun, currently), locate any required libraries and object files, and place the newly compiled object files and the executable in a different directory for each architecture.

Once it is verified that the changes resulted in the desired effect, and no new problems have been introduced, the modules should be checked in and the copies in the project directory resynchronized. Make should then be invoked from the project directory for each machine, followed by "make release" to place the new executable (and any include files and object files needed by other projects) where the production scripts and other makefiles can find them.

See the comments in the makefile for more specific information, such as how to change code involved in reading/writing netCDF files.

Appendix A, Auxiliary Programs

The Fortran subroutine read_vec provides the ability to read a vector file and is available for general distribution. It should be updated whenever the format or content of the vector output

file is changed. The program tstread_vec provides an example and exercise of its use, and should also be maintained.

The IDL program vecview.pro will take an input vector file and show a color plot of the meridional and zonal winds as a function of altitude and track angle. It is intended for use in daily production.

The perl script runVector.pl is available for invoking vector in production. Reference (10) provides more details.