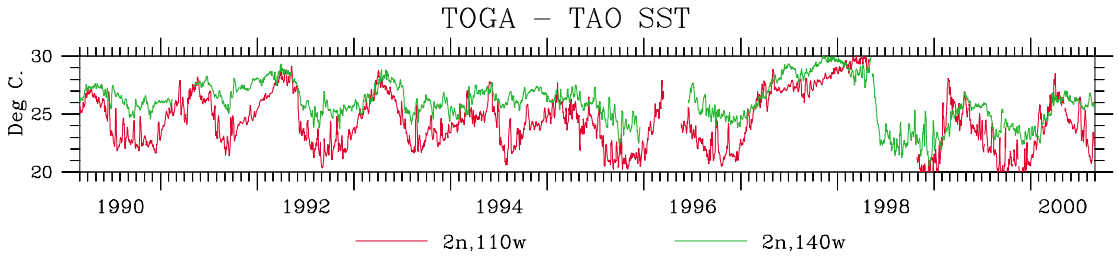
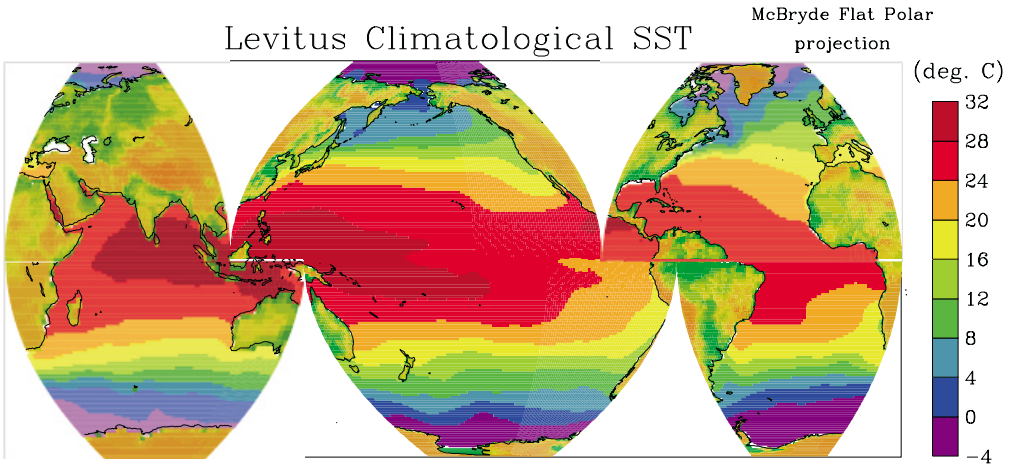


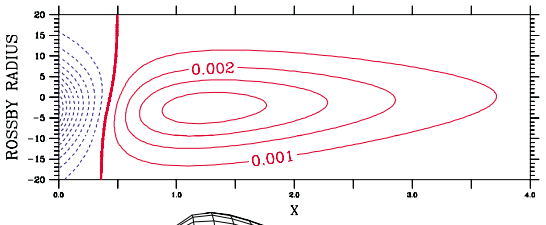

# NOAA / PMEL

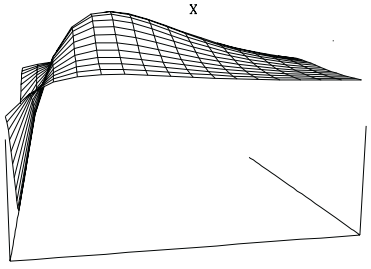
# F E R R E T

## An Analysis Tool for Gridded Data



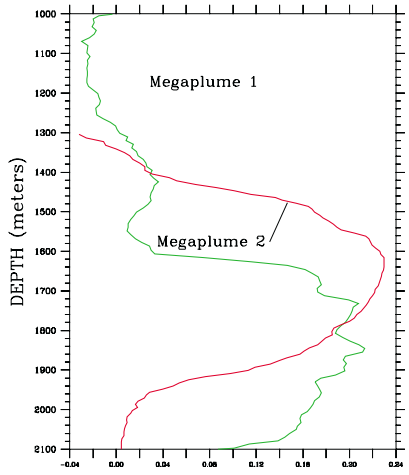

Perturbation Solution



$-(B+1/2)*D_y*e^B/D^2$

VENTS Megaplume Thermal Structure



# FERRET

# USER'S GUIDE

## Version 5.22

NOAA/PMEL/TMAP

Steve Hankin
Jon Callahan, Ansley Manke
Kevin O'Brien, Joe Sirott
September 6, 2000

# About the Cover

The cover of this User's Guide was produced by Ferret. From the top down the plots are: "TOGA-TAO SST," time series from the Tropical Pacific TAO array; "Levitus Climatological SST," an equal area projection of level one of the annual Climatological Atlas of the World Oceans by Sydney Levitus of NOAA/NODC; "Perturbation Solution," a visualization of abstract functions by Dr. Ping Chang; "Vents Megaplume Thermal Structure," vertical temperature profiles of undersea thermal vents from the NOAA Vents program.

# Contents

## CHAPTER 1: INTRODUCTION



## CHAPTER 2: DATA SET BASICS

**CHAPTER 3: VARIABLES AND EXPRESSIONS**

## CHAPTER 4: GRIDS AND REGIONS



## CHAPTER 5: ANIMATIONS AND GIF IMAGES

## CHAPTER 6: CUSTOMIZING PLOTS

**CHAPTER 7: HANDLING STRING DATA: "SYMBOLS"**



**CHAPTER 8: WORKING WITH SPECIAL DATA SETS**



**CHAPTER 9: COMPUTING ENVIRONMENT**

**CHAPTER 10: CONVERTING TO NETCDF**



**CHAPTER 11: WRITING EXTERNAL FUNCTIONS**

**PART II: COMMANDS REFERENCE**

# Chapter 1: INTRODUCTION

## Ch1 Sec1. OVERVIEW

Ferret is an interactive computer visualization and analysis environment designed to meet the needs of oceanographers and meteorologists analyzing large and complex gridded data sets. "Gridded data sets" in the Ferret environment may be multi-dimensional model outputs, gridded data products (e.g., climatologies), singly dimensioned arrays such as time series and profiles, and for certain classes of analysis, scattered n-tuples (optionally, grid-able using Ferret's objective analysis procedures). Ferret accepts data from ASCII and binary files, and from two standardized, self-describing formats. Ferret's gridded variables can be one to four dimensions—usually (but not necessarily) longitude, latitude, depth, and time. The coordinates along each axis may be regularly or irregularly spaced

Ferret offers the ability to define new variables interactively as mathematical expressions involving data set variables and abstract coordinates. Calculations may be applied over arbitrarily shaped regions. Ferret's "external functions" framework allows external code written in FORTRAN, C, or C++ to merge seamlessly into Ferret at runtime. Using external functions, users may easily add specialized model diagnostics, advanced mathematical capabilities, and custom output formats to Ferret. A collection of general utility external functions is included with Ferret.

Ferret provides fully documented graphics, data listings, or extractions of data to files with a single command. Without leaving the Ferret environment, graphical output may be customized to produce publication-ready graphics. Graphic representations include line plots, scatter plots, line contours, filled contours, rasters, vector arrows, polygonal regions and 3D wire frames. Graphics may be presented on a wide variety of map projections. Interfaces to integrate with 3D and animation applications, such as Vis5D and XDataSlices are also provided.

Ferret has an optional point-and-click graphical user interface (GUI). The GUI is fully integrated with Ferret's command line interface. The user may freely mix text-based commands with mouse actions (push buttons, etc.). Ferret's journal file will log all of the actions performed during a session such that the entire session, including GUI inputs, can be replayed and edited at a later time.

This User's Guide describes only the command line interface to Ferret. Other documents describe the point and click interface.

Ferret was developed by the Thermal Modeling and Analysis Project (TMAP) at NOAA/PMEL in Seattle to analyze the outputs of its numerical ocean models and compare them with gridded, observational data. Model data sets are often multi-gigabyte in size with mixed 3- and 4-dimensional variables defined on staggered grids.

Ferret is supported on a variety of Unix workstations with a version also available for Windows NT/9x. Ferret is available at no charge from anonymous FTP [node ftp.ferret.noaa.gov] or from the World Wide Web [URL http://ferret.wrc.noaa.gov/Ferret].

## Ch1 Sec1.1. Ferret User's Group

The Ferret User's Group provides a venue to ask experienced Ferret users for advice solving problems and to keep abreast of the latest Ferret updates. To (un)join simply send an e-mail message to

```
Majordomo@ferret.wrc.noaa.gov
```

and include a message which says simply

```
(un)subscribe ferret_users
```

(Note this must be in the e-mail message BODY—not in the subject line.) To learn about the user's list without joining send this message instead to the same address:

```
info ferret_users
```

## Ch1 Sec1.2. Ferret Home Page

The Ferret Home Page contains source code distributions, on line documentation, Users' Group archives, Frequently Asked Questions and more. It is available at

```
http://ferret.wrc.noaa.gov/Ferret/FAQ
```

## Ch1 Sec2. GETTING STARTED

A quick way to get to know Ferret is to run the tutorial provided with the distribution.

```
% ferret
yes? GO tutorial
```

If Ferret is not yet installed consult the chapter "Computing Environment" (p. 185). (The tutorial is also available through the World Wide Web through Ferret's on-line demonstrations page..) The tutorial demonstrates many of Ferret's features, showing the user both the commands given and Ferret's textual and graphical output. You may find the explanations, terms and examples in this manual easier to understand after running the tutorial.

## Ch1 Sec2.1. Concepts

Words in bold below are defined in the glossary of this manual.

In Ferret all **variables** are regarded as defined on **grids**. The grids tell Ferret how to locate the data in space and time (or whatever the underlying units of the grid **axes** are). A collection of variables stored together on disk is a **data set**

To access a variable Ferret must know its name, data set and the **region** of its grid that is desired. Regions may be specified as **subscripts** (indices) or in **world coordinates** Data sets, after they have been pointed to with the SET DATA command (alias "USE"), may be referred to by data set number or name.

Using the LET command new variables may be created "from thin air" as **abstract expressions** or created from combinations of known variables as arbitrary **expressions** If component variables in an expression are on different grids, then **regridding** may be applied simply by naming the desired grid.

The user need never explicitly tell Ferret to read data. From start to finish the sequence of operations needed to obtain results from Ferret is simply:

1) specify the data set
2) specify the region
3) define the desired variable or expression (optional)
4) request the output

For example (Figure 1_1),

```
yes? USE coads              !global sea surface data
yes? SET REGION/Z=0/T="16-JAN-1982"/X=160E:160W/Y=20S:20N
yes? VECTOR uwnd,vwnd       !wind velocity vector plot
```

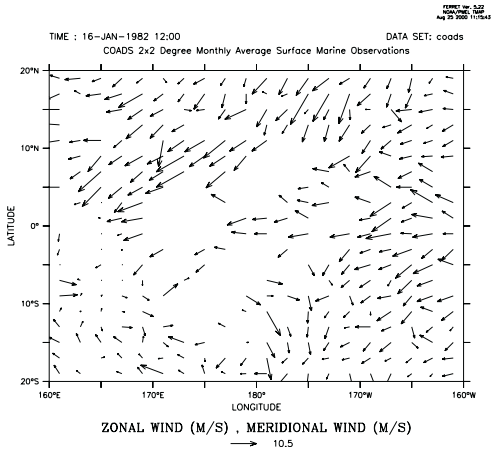



Figure 1_1

## Ch1 Sec2.1.1. Thinking like a Ferret:

(A discussion on the Ferret outlook on the concepts of data, variables, grids and other basics of Ferret.)

**Plottable variables**

For this discussion we will coin the term "plottable variables." There are no non-plottable variables that will come up in this discussion but "variables" is a bit too generic. Plottable variables are of 3 types:

- file variables – read from disk files
- user-defined variables – defined by the LET command
- pseudo-variables – regions (I,J,K,L,X,Y,...) used as variables

As much as possible Ferret tries to make all types of variables indistinguishable. All plottable variables are defined on **grids**. No plottable variables exists in a vacuum for Ferret. The grid on which a plottable variable exists tells how to locate the variable in space and time. In cases where the variables are abstract in nature—disconnected from space and time—Ferret will associate those variables with grids that are abstract, too. Where a geographical grid will associate the Nth position along an axis with a location (like 20 degrees north latitude) an abstract grid will simply associate the Nth position with the number N. Plottable variables may be regridded to other grids than the one on which they are defined. (Done with "G=".)

All references to plottable variables must have a complete **context**. A complete context will be described in detail later—briefly it means a region in space, an interval in time and the data set(s) in which the variables will be found.

**Grids**

All Ferret grids are 4-dimensional. In most cases the axes have the obvious interpretation of 3 space coordinates and time but sometimes the axes are abstract.

A grid is composed of 4 axes, each describing the coordinates along one dimension. 3d, 2d, 1d and 0d grids are regarded as special cases of the full 4 dimensions in which 1 or more axes are set to "NORMAL."

Ferret tries to look at all axes equally—the same syntax of regions and transformations applies to each. Calendar dates, east-west longitudes and north-south latitudes are merely convenient ways to format positions along axes that have special interpretations to people—not to Ferret. (The only exception to this is that if the Y axis has units of Latitude Ferret will insert cosine(Latitude) factors into some calculations.)

Axes and grids may be defined by "grid files" (which normally have .GRD filename extensions). Axes may also be defined by the DEFINE AXIS command; grids by the DEFINE GRID command.

## Contexts

A **context** is a region or point in space and time and a data set(s). This is the information needed by Ferret to make sense of a reference to a plottable variable. Suppose that "U" is a variable in a data set (file) called U_DATA. A command like "PLOT U" is meaningful only when Ferret knows that it is supposed to be looking for U in data set U_DATA and knows where in 4-dimensional space it is supposed to plot.

The context **space-time region** may be described by a mix of subscript and world coordinate positions. **Subscripts** are specified by I=,J=,K=,L= for axes 1 through 4, respectively. **World coordinates** are specified by X=,Y=,Z=,T=. On the right of the equal sign a single point may be given or a range specified by low:high may be given. Special formats are allowed for X= (longitude, e.g. 160W), Y=(latitude, e.g. 23.5S) and time (calendar dates like "7-NOV-1989:12:35:00" in quotation marks).

The data set may be given by name or number. The commands SET DATA and CANCEL DATA and the D= context descriptor all accept the name of the data set or its number. The data sets are numbered by the order in which they are pointed to with SET DATA. This order may be seen with SHOW DATA.

You can tell Ferret the context in 3 places:

1. The program context: Using the commands SET REGION and SET DATA you can describe a context in which all commands and expressions will be interpreted. You can look at the program context with SHOW REGION and SHOW DATA. (The command SET DATA is used both to initialize new data sets and to make previously initialized sets the current program context. When SET DATA initializes a new data set that set automatically becomes the data set for the program context.) Example: SET REGION/Z=50
2. The command context: Using the command qualifiers I,J,K,L,X,Y,Z,T and D commands like PLOT,CONTOUR,SHADE,LIST and VECTOR can specify additional context information. Command context information on any axis or on the data set will replace any program context information on the same axis or the data set.
3. The variable context: Using the same qualifiers as the command context any plottable variable name can be modified with additional context information in square brackets (e.g. U[Z=200,D=U_DATA]). Variable context information on any axis or the data set will replace any program or command context information on the same axis or the data set.

## Transformations

Ferret can **transform** plottable variables along their axes. Transformations may be specified only in the variable context. Ferret understands a number of transformations that may be specified with the space-time region qualifiers. Some examples: PLOT U[Z=0:100@AVE] — the variable U averaged between Z=0 and Z=100 LIST/L=1:200 U[L=@SBX:5] — U with a boxcar smoother of width 5 points along L.

Also,

- @FAV (fill data holes with averages)
- @DIN (definite integral) @IIN (indefinite integral)
- @DDC (centered derivative)
- @SHF (shift data a number of points along an axis)
- @MIN (minimum value along an axis)

... and others (see HELP TRANSFORMATIONS inside Ferret)

## Ch1 Sec2.2. Unix command line switches

```
ferret [-batch<file>.ps][-memsize Mwords] [-unmapped] [-gui] [-help]
```

**-memsize Mwords**
specify the memory (data cache) size in Megawords default: 3.2

**-unmapped**
use invisible output windows (useful for creating animations and GIF files)

**-gui**
start Ferret in point-and-click mode (may not be available on all platforms)

**-help**
obtain help on the Unix command line options

**-gif**
Ferret can run in batch mode—without an X server. Graphical output is buffered, and is stored in a GIF file by executing the FRAME command. For example:

```
yes? FRAME/FILE=picture.gif
```

sends the stored graphical output from Ferret to the GIF file picture.gif.

Please note the following when using batch mode:

- Window resizing only works if the window is cleared before resizing the window. For instance:
  ```
  yes? set window/clear/size=0.25
  ```
  will resize the window while
  ```
  yes? set window/size=0.25/clear
  ```
  will cause an error.
- Avoid metafile commands when running in batch mode. In particular,
  ```
  yes? set mode meta
  ```
  may cause problems.
- Don’t create new Ferret windows when running without an X server. The following command:

```
yes? set window/new
```
will cause Ferret to crash.

**-batch**

Ferret can generate PostScript files without an X server. If you wish to use this mode, start Ferret with the -batch option:

```
ferret -batch <file>.ps
```

where <file> is the name of the output file. Note that the filename must end with ".ps".

Please note the following when using PostScript mode:

- The PostScript output will not be fully written to the output file until you exit from Ferret.
- Window sizing commands do not have any effect on PostScript output.
- Avoid metafile commands when running in PostScript mode.
- Don't create new Ferret windows when running without an X server. The following command:

  ```
  yes? set window/new
  ```

  will cause Ferret to crash.

## Ch1 Sec2.3. Sample sessions

This section presents a number of short Ferret sessions that demonstrate common uses. Data sets used in these sessions and throughout this manual are included with the distribution. If Ferret is installed on your system, you can duplicate the examples shown.

### Ch1 Sec2.3.1. Accessing a NetCDF data set

In this sample session, the data set "monthly_navy_winds" is specified and certain aspects of it are examined. The command SHOW DATA/VARIABLES displays the variables in "monthly_navy_winds" and where on each axis they are defined. SET REGION specifies where in the grid the user wishes to examine the data. VECTOR produces a vector plot of the indicated variables over the specified region.

```
yes? USE monthly_navy_winds            ! specify the data set
yes? SHOW DATA/VARIABLES              ! what's in it?
     currently SET data sets:
    1> /opt/local/ferret/fer_dsets/descr/monthly_navy_winds.des
(default)
     FNOC 2.5 Degree 1 Month Average World-wide Wind Field
 name     title                          I         J         K
L
 UWND     ZONAL WIND                    1:144     1:73      ...
1:132
            M/S on grid FNOC251 with -99.9 for missing data
```

```
            X=18.8E:18.8E(378.8)  Y=91.2S:91.2N
 VWND     MERIDIONAL WIND              1:144     1:73      ...
1:132
            M/S on grid FNOC251 with -99.9 for missing data
            X=18.8E:18.8E(378.8)  Y=91.2S:91.2N
  time range: 16-JAN-1982 20:00 to 17-DEC-1992 03:30
```

## Ch1 Sec2.3.2. Reading an ASCII data file

Many examples of accessing ASCII data are available later in this manual. See the chapter, “Data Sets” (p. 29) The simplest access, one variable with one value per record, looks like this:

```
% ferret
yes? FILE/VARIABLE=v1 snoopy.dat
yes? PLOT v1
yes? QUIT
```

## Ch1 Sec2.3.3. Using viewports

The command SET VIEWPORT allows the user to divide the output graphics “page” into smaller display viewports.

In this sample session, we create two plots in two halves of a window (Figure 1_2):

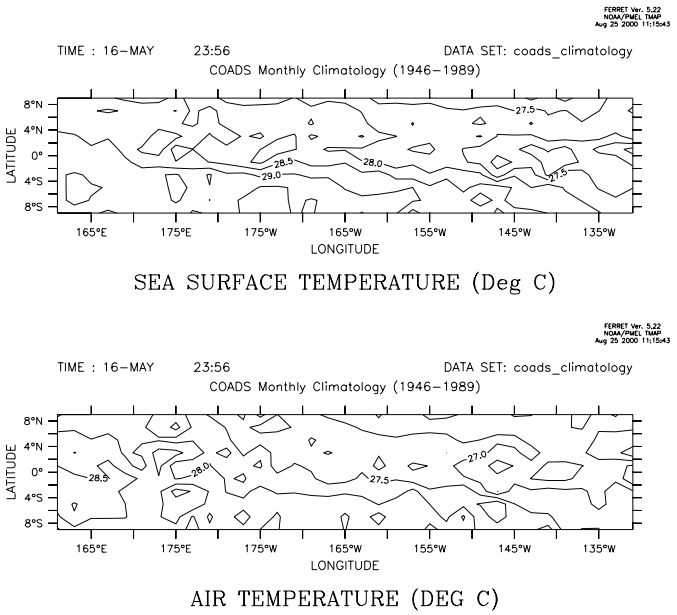



Figure 1_2

```
% ferret
yes? USE coads_climatology
yes? SET REGION/X=160E:130W
yes? SET REGION/Y=-10:10/L=5
yes? SET VIEWPORT upper
```

```
yes? CONTOUR sst
yes? SET VIEWPORT lower
yes? CONTOUR airt
yes? QUIT
```

## Ch1 Sec2.3.4. Using abstract variables

Abstract variables (expressions that contain no dependencies on disk-resident data) can be easily displayed with Ferret. See the chapter "Variables and Expressions", section "Abstract variables" (p. 50), for several examples and detailed information.

For example, a user wishing to examine the function SIN(X) on the interval [0,3.14] might use (Figure 1_3):

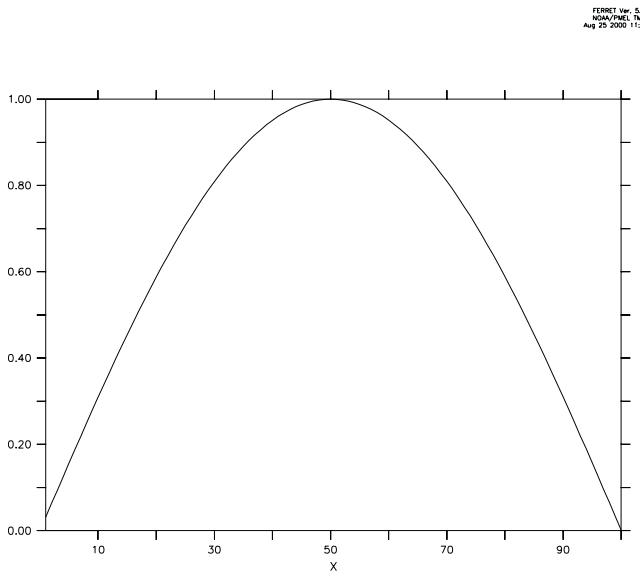



Figure 1_3

```
% ferret
yes? PLOT/I=1:100 sin(3.14*I/100)
yes? QUIT
```

## Ch1 Sec2.3.5. Using transformations

A transformation is an operation performed on a variable along a particular axis and is specified with the syntax "@trn" where "trn" is the name of a transformation. See the chapter "Variables and Expressions", section "Transformations" (p. 75), for detailed information.

A user may wish to look at ocean temperatures averaged over a range of depths. In this sample session, we look at temperatures averaged from 0 to 100 meters of depth using a data set which

has detailed resolution in depth (Figure 1_4). We plot the data along longitude 160 west from latitude 30 south to 30 north.

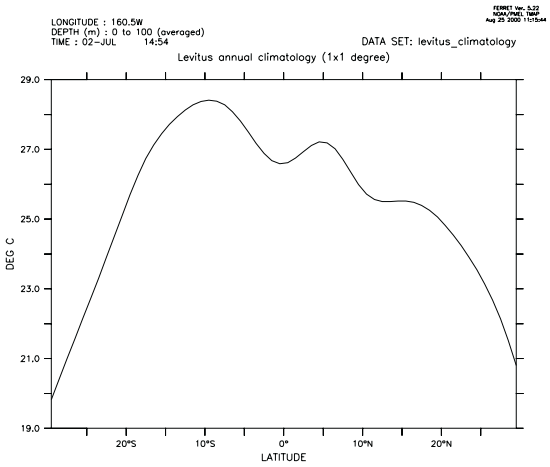



Figure 1_4

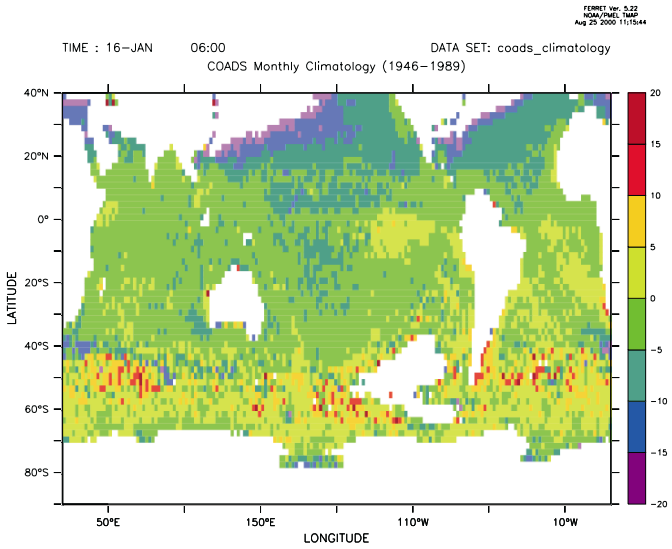
```
% ferret
yes? USE levitus_climatology
yes? SET REGION/Y=30s:30n/X=160W
yes? PLOT temp[Z=0:100@AVE]
yes? QUIT
```

## Ch1 Sec2.3.6. Using algebraic expressions

See the chapter “Variables and Expressions”, section “Expressions” (p. 53) for a description of valid expressions.

In this example, the data set contains raw sea surface temperatures, air temperatures, and wind

speed measurements. We wish to look at a shaded plot of sensible heat at its first timestep (L=1) (Figure 1_5). We specify a latitude range and contour levels.




SENSIBLE HEAT

Figure 1_5

```
% ferret
yes? USE coads_climatology              !monthly COADS climatology
yes? LET kappa = 1                      !arbitrary
yes? LET/TITLE="SENSIBLE HEAT"  sens_heat = kappa * (airt-sst) * wspd
yes? SHADE/L=1/LEV=(-20,20,5)/Y=-90:40 sens_heat
yes? QUIT
```

## Ch1 Sec2.3.7. Finding the 20-degree isotherm

Isotherms can be located with the "@LOC" transform, which returns the axis location where the value of the argument of @LOC first occurs. Thus, "TEMP[Z=0:200@LOC:20]" locates the first occurrence of the temperature value 20 along the Z axis, scanning all the data between 0 and 200 meters.

A session examining the 20-degree isotherm in mid-Pacific ocean data (Figure 1_6):

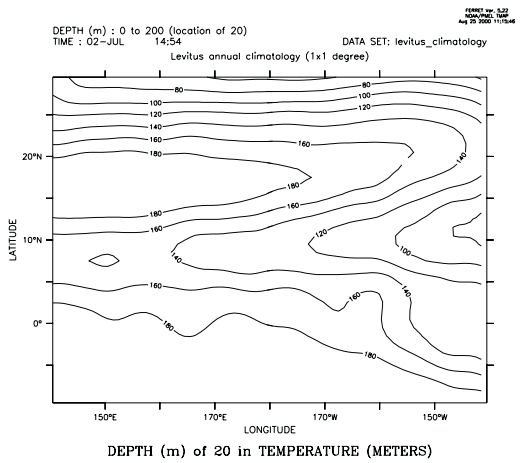



Figure 1_6

```
% ferret
yes? USE levitus_climatology
yes? SET REG/Y=10s:30n/X=140E:140W
yes? PPL CONSET .12  !label size
yes? CONTOUR temp[Z=0:200@LOC:20]
yes? QUIT
```

Note that the transformation @WEQ could have been used to display ANY variable on the surface defined by the 20 degree isotherm.

## Ch1 Sec3. COMMON COMMANDS

A quick reference to the most commonly used Ferret commands (typing "SHOW COMMANDS" at the Ferret prompt lists all commands):

| **Command** | **Description** |
|---|---|
| USE | names the data set to be analyzed (alias for "SET DATA") |
| SHOW DATA | produces a summary of a variable |
| SHOW GRID | examines the coordinates of a grid |
| SET REGION | sets the region to be analyzed |
| LIST | produces a listing of data |
| PLOT | produces a plot |
| CONTOUR | produces a line contour plot |
| FILL | produces a color filled contour plot |
| SHADE | produces a shaded-area plot |
| VECTOR | produces a vector arrow plot |
| POLYGON | plots polygonal regions |

| Command | Description |
|---|---|
| DEFINE | define new axes, grids, and symbols |
| STATISTICS | produces summary statistics about variables and expressions |
| LET | defines a new variable |
| SAVE | saves data in NetCDF format |
| GO | executes Ferret commands contained in a file |

Information on all Ferret commands is available in Part II, Commands Reference, of this manual.

## Ch1 Sec4. COMMAND SYNTAX

Commands in program Ferret conform to the following template:

```
COMM [/Q1/Q2...] [SUBCOM[/S1/S2...]] [ARG1 ARG2 ...] [!comment]
```

where

| | |
|---|---|
| COMM | is a command name `yes? LIST` |
| Q1... | are qualifiers of the command `yes? CONTOUR/SET_UP` |
| SUBCOM | is a subcommand name `yes? SHOW MODE` |
| S1... | are qualifiers of the subcommand `yes? SET LIST/APPEND` |
| ARG1... | are arguments of commands `yes? CANCEL MODE INTERPOLATE` |

notes...

- Items in square brackets are optional.
- One or more spaces or tabs must separate the command from the subcommand and from each of the arguments. Spaces and tabs are optional preceding qualifiers.
- Multiple commands, separated by semi-colons, can be given on the same line.
- Command names, subcommand names, and qualifiers require at most 4 characters. (e.g., `yes? CANCEL LIST/PRECISION` is equivalent to `yes? CANC LIST/PREC`)
- Some qualifiers take an argument following "=" (e.g., `yes? LIST/Y=10S:10N`).
- An exclamation mark normally signifies the end of a command and the start of (optional) comment text.
- The backslash character (\), when placed directly before an exclamation point (!), apostrophe (‘), semicolon (;), or forward slash (/), will hide it ("escape it") from Ferret.
- See the Expressions section (p. 53) for information on algebraic expressions as arguments to commands
- See the Symbols sections (p. 167) for information on symbol substitution in commands

## Ch1 Sec5. GO FILES

GO files are files containing Ferret commands. They can be executed with the command "GO filename". Throughout this manual, these files are referred to as GO scripts or journal files (the file names end in *.jnl). There are two kinds of GO files provided with the distribution (differing in function, not form)—demos and tools. A list of the demonstrations and scripts can be found in Ferret's on-line documentation in "on-line demonstrations".

## Ch1 Sec5.1. Demonstration files

Demonstration GO files provide examples of various Ferret capabilities (the tutorial is such a script) . The demonstration GO files may be executed simply by typing the Ferret command

```
yes? GO demo_name
example:  yes? GO vector_demo
```

Below is a list of the demo files provided as of 4/99 (located in directory $FER_DIR/examples). The Unix command "Fgo demo" will list all GO scripts containing the string "demo". Use Fgo '*' to see all the scripts that are currently available on your system.

| Name | Description |
|---|---|
| tutorial | brief tour through Ferret capabilities |
| topographic_relief_demo | global topography |
| coads_demo | view of global climate using the Comprehensive Ocean-Atmosphere Data Set |
| levitus_demo | T-S relationships using Sydney Levitus' climatological Atlas of the World Oceans |
| fnoc_demo | Naval Fleet Numerical Oceanography Center data |
| vector_demo | vector plots |
| wire_frame_demo | 3D wire frame representation |
| custom_contour_demo | customized contour plots |
| viewports_demo | output to viewports |
| multi_variable_demo | multiple variables with multiple dependent axes |
| objective_analysis_demo | interpolating scattered data to grids |
| mp_demo | map projections demo |
| log_plot_demo | log plots using PPLUS in Ferret |
| depth_to_density_demo | contour with a user-defined variable as an axis |
| file_reading_demo | reading an ASCII file |
| regridding_demo | tutorial on regridding data |
| mathematics_demo | abstract function calculation |
| statistics_demo | probability distributions |
| spirograph_demo | for-fun plots from abstract functions |
| splash_demo | for-fun mathematical color shaded plots |
| symbol_demo | how to use symbols for plot layouts |

| Name | Description |
|---|---|
| sigma_coordinate_demo | how to work with sigma coordinates |

## Ch1 Sec5.2. GO tools

GO tools are scripts which contain Ferret commands and perform dataset-independent tasks. For example, “GO land” overlays the outline of the continents on your plot. (**Note:** In order for Ferret to locate the GO scripts, the environment variable FER_GO must be properly defined. See the chapter “Computing Environment,” p. 185, for guidance.)

To run any GO tool, from the Ferret command line, type,

```
Yes? GO scriptname
```

Or if the script has arguments, they follow the script name with optional comma separators.

```
yes? GO script2 arg1, arg2
```

To find out about the script, use the /HELP qualifier, which opens the script with the more command to type the first 20 lines of the script and allow you to see the documentation at the start of the script.

```
yes? GO/HELP scriptname
```

To omit arguments from a GO script,

```
yes? GO script arg1, , arg3
```

Or double quotes with a space to indicate the missing item.

```
yes? GO script arg1 "  " arg3
```

The Unix command Fgo has been provided to assist with locating tools within the Unix directory hierarchy. For example,

| | |
|---|---|
| `% Fgo grid` | displays all tools with the substring “grid” in their names |
| `% Fgo '*'` | displays all GO tools and demonstrations |

When passing arguments to GO commands sometimes it is necessary to pass enclosing quotation marks. An common example is the passing of the argument to the CONTOUR/LEVELS qualifier in cases such as

```
CONTOUR/LEVELS="(-100) (-10,10,2) (100)" my_var
```

where there may be blanks embeddd inside of the string. There are 3 methods to embed quotations inside of strings

1. use "\" to protect the quotation marks in the GO command line

```
yes? go my_go_script "\"(-100) (-10,10,2) (100)"\"
```

with the script containing the line

```
CONTOUR/LEVELS=$1 my_var
```

2. use "\" to define a symbol which contains the quotation marks

```
yes? DEFINE my_quoted_string \"$1\"
yes? CONTOUR/LEVELS=($my_quoted_string) my_var
```

3. use the symbol substitution syntax to add quotes to theGO argument

```
Yes? CONTOUR/LEVELS=$1&|*>"*"&
```

Of course, in the above examples one could also simply use

```
yes? CONTOUR/LEVELS="$1" my_var
```

Below is a table of the tools provided with your Ferret installation. Some tools accept optional arguments to control details. Use `Fgo -more script_name` for details on a script.

| Tool name | Description |
|---|---|
| OVERLAYS | |
| basemap | a geographical basemap of continents to overlay on |
| land | overlays continental boundaries (color controls) |
| bold_land | overlays darker continental boundaries |
| fland | overlays filled continents (color and resolution controls) |
| focean | overlays ocean mask (for terrestrial plots) |
| graticule | sets the plot axis style to use a graticule (rather than tics) |
| tics | resets the plot style to use axis tics (rather than a graticule) |
| gridxy | overlays a “graticule” labeling the I,J subscripts |
| gridxz | overlays a “graticule” labeling the I,K subscripts |
| gridxt | overlays a “graticule” labeling the I,L subscripts |

| Tool name | Description |
|---|---|
| gridyz | overlays a “graticule” labeling the J,K subscripts |
| gridyt | overlays a “graticule” labeling the J,L subscripts |
| gridzt | overlays a “graticule” labeling the K,L subscripts |
| box | draws a box at the specified location on the plot |
| ellipse | draws an ellipse at the specified location on the plot |
| MATHEMATICAL | |
| frequency_histogram | makes a frequency distribution plot (histogram) of data |
| ts_frequency | creates a 2-variable histogram (typically an oceanographer’s TS density diagram) |
| polar | defines R and THETA from X and Y to perform (limited) polar plots |
| regressx | defines variables for linear regression along X axis |
| regressy | defines variables for linear regression along Y axis |
| regressz | defines variables for linear regression along Z axis |
| regresst | defines variables for linear regression along T axis |
| unit_square | sets unit square as default for abstract variables |
| variance | defines variables to compute variances and covariances |
| var_n | refines TVARIANCE with corrected n/n+1 factors |
| dynamic_height | defines Ferret variables for dynamic height calculations |
| SAMPLE DISPLAYS | |
| line_samples | draws specimens of the available line styles |
| line_thickness | draws examples of pen color/thickness styles in PPLUS |
| fill_samples | draws specimens of the available fill styles |
| show_symbols | draws specimens of the default symbols |
| show_88_syms | draws specimens of all 88 PPLUS symbols |
| GRAPHICS | |
| bar_chart | makes a color-filled bar chart from a line of data |
| bar_chart2 | makes a bar chart using hollow rectangles |
| centered_vectors | makes a vector plot with coords at vector midpoints |
| scattered_vectors | makes a vector plot from an ASCII file: x,y,u,v |
| stick_vectors | makes a stick vector plot of a line of U,V values |
| extremum | annotate contour extrema on a plot |
| split_z | oceanographic-style plot with 2 z-axis scalings |
| PLOT APPEARANCE | |
| margins | tweak the sizing of the plot on the page |
| magnify [factor] | increases the data plotting area (area inside the axes) |
| unmagnify | restores the plot origin and axis lengths to default values |
| black | sets video background to black, foreground to white |
| white | sets video background to white, foreground to black |
| bold | sets up PLOT+ and Ferret to produce bolder-looking plots |

| Tool name | Description |
|---|---|
| unbold | resets plot environment to normal after “GO bold” |
| unlabel [label #] | removes a specified (numbered) PPLUS movable label |
| remove_logo | removes labels 1–3 that form the Ferret logo |
| box_plot | produces a plot with “bare” axes (no tics, no labels) |
| reminder | place small annotations in upper left corner of plot |
| COLOR | |
| try_palette [pal] | displays palette appearance for various numbers of color levels |
| try_centered_palette | displays centered palette appearance for various numbers of levels |
| exact_colors | sets up Ferret and PPLUS to modify individual colors in a color palette |
| squeeze_colors | modifies a color palette by squeezing and stretching the color scale |
| MULTIPLE X AND Y AXES (run demo: `yes? GO multi_variable_plots`) | |
| left_axis_plot | plots a single variable preparing for a 2nd axis on the right |
| right_axis_plot | overlays a plot of a single variable using an axis on the right |
| multi_xaxis_plot1 | draws a plot formatted for later overlays using multiple X axes |
| multi_xaxis_overlay | overlays a variable with a distinct X axis |
| multi_yaxis_plot1 | draws a plot formatted for later overlays using multiple Y axes |
| multi_yaxis_overlay | overlays a variable with a distinct Y axis |
| MAP PROJECTIONS (run demo: `yes? GO mp_demo`) | |
| mp_~name~ | individual projections include bonne, craster_parabolic, eckert_greifendorff, eckert_iii, eckert_v, hammer, lambert_cyl, mcbryde_fpp, mercator, orthographic, plate_caree, polyconic, sinusoidal, stereographic_eq, stereographic_north, stereographic_south, vertical_perspective, wagner_vii, winkel_i |
| mp_aspect | set the appropriate window aspect ratio for this map projection |
| mp_fland | overlays “map projected” filled continents (color controls) |
| mp_graticuled | overlays “map projected” graticule (color controls) |
| mp_label | plots a label using world coordinates |
| mp_land | overlays “map projected” continental boundaries (color controls) |
| mp_land_stripmap | creates a land-centric, interrupted “stripmap” using the current map projection |
| mp_line | overlays “map projected” plotted data |
| mp_ocean_stripmap | creates an ocean-centric, interrupted “stripmap” using the current map projection |
| mp_polygon | overlays “map projected” polygons |
| SAMPLING A GRIDDED FIELD | |
| bullseye | locate a bullseye in a 2d field |
| digitize | obtain data values from a plot using the cursor |

| Tool name | Description |
|---|---|
| TESTS | |
| test | tests proper functioning of FER_GO |
| ptest | produces a quick test plot |
| squares | creates a filled-area test plot |

## Ch1 Sec5.3. Writing GO tools

A GO tool ("GO script," "journal file," ...) is simply a sequence of Ferret commands stored in a file and executed with the GO command. Writing a simple GO tool requires nothing more than typing normal commands into a file.

To write a robust GO tool that may be shared, however, certain guidelines should be followed:

1) the GO tool should be well documented
2) the GO tool should leave the Ferret context unmodified
3) the GO tool may need to run "silently"
4) the GO tool may need to accept arguments (parameters)

### Ch1 Sec5.3.1. Documenting GO tools

Documentation consists primarily of well-chosen comment lines (lines beginning with an exclamation mark). In addition, a line of this form should be included:

```
! Description:  [one-line summary of your GO tool]
```

This line is displayed by the Fgo tool.

### Ch1 Sec5.3.2. Preserving the Ferret state in GO tools

Often a complex GO tool requires setting data sets, modifying the current region, etc. But to a user executing this tool its behavior may seem erratic if the user's previous context is modified by running the tool. A tool can restore the previous state of Ferret by these means:

region: Save the current default region with the command DEFINE REGION/DEFAULT save. Restore it at the end of your GO tool with SET REGION save.

data set: Save the current default data set with SET DATA/SAVE. Restore it at the end of your GO tool with SET DATA/RESTORE.

grid: Save the current default grid set with SET GRID/SAVE. Restore it at the end of your GO tool with SET GRID/RESTORE.

modes: If you modify a mode inside your GO tool by issuing a SET MODE or a CANCEL MODE command the original state of that mode can be restored using SET MODE/LAST.

## Ch1 Sec5.3.3. Silent GO tools

If a user has set mode "verify" then by default every line of your GO tool, including comment lines, will be displayed at the screen as Ferret processes it. To make your GO tool run silently include the command CANCEL MODE VERIFY at the beginning of the GO tool and SET MODE/LAST VERIFY at the end. If the backslash character "\" is found at the beginning of any line that single line will not be displayed regardless of the state of MODE VERIFY. Thus the command "\CANCEL MODE VERIFY" is often the first line of a GO tool. Note also that the command LET/SILENT is useful in GO tools which need to define variables.

## Ch1 Sec5.3.4. Arguments to GO tools

Arguments (parameters) may be passed to GO tools on the command line. For example,

```
yes? GO land red
```

passes the string "red" into the GO file named land.jnl. Inside the GO tool the argument string "red" is substituted for the string "$1" wherever it occurs. The "1" signifies that this is the first argument—similar logic can be applied to $1,... $9 or $0 where $0 is replaced by the name of the GO tool itself. Similarly "$*" is replaced by all the arguments, 1–9 as a single string, separated by spaces.

As Ferret performs the substitution of $1 (or other) arguments it offers a number of string processing and error processing options. For example, without these options, if a user failed to supply an argument to "GO land" then Ferret would not know what to substitute for $1 and it would have to issue an error message. A default value can be supplied by the GO tool writer using the syntax

```
$1%string%
```

for example,

```
$1%black%
```

inside land.jnl would default to “black” if no color were specified. Note that in the example percent signs were used to delimit the default string but any of the characters ! # $ % or & also work as delimiters.

In another case it might not be appropriate to supply a default string but instead it would be desirable to issue an instructional error message. The “<” character indicates an error message text:

```
$1"<you must supply an argument to this GO tool"
```

In still other cases there are a range of acceptable arguments but all other arguments are illegal. The allowable arguments can be specified following “|” (vertical bar) characters as in this example:

```
$1"|black|red|<You must specify black or red"
```

or a default of “black” could be specified together with the options as

```
$1"black|black|red|"
```

In the interest of “friendliness” a GO file may want to allow the user to specify a string other than the string actually needed by the GO tool. For example, a red plot line is actually obtained by the PLOT command qualifier /LINE=2—the string “red” never appears in this command. To allow a user to specify “red” and yet have the string “2” substituted, Ferret has provided the replacement arrow “>”. Thus

```
$1"1|red>2|"
```

specifies a default string of “1” if no argument is given but substitutes “2” if “red” is supplied. In a typical GO tool line, defaults, options, substitutions, and an error message are combined like this:

```
PLOT/LINE=$1"1|red>2|green>3|blue>4|<must be red, green, or blue"
```

Note that the error message will be issued only if some color other than “red,” “green,” or “blue” is specified; if no argument is specified then “1” is substituted.

An asterisk (*) can be used to designate that any text whatsoever is acceptable as an option.

```
PLOT/LINE=$1"1|red>2|green>3|blue>4|*>7"
```

would never generate an error and would use line style 7 (thick black) if an unrecognized argument string such as “orange” were given.

An asterisk (*) can also be used on the right-hand side of a substitution, in which case it stands for the entire original argument string. For example

```
SET VARIABLE/TITLE=$1%*>"*"%
```

will place double quotation marks around the string in argument 1.

A final style note to keep in mind when writing GO tools that use arguments: providing error message feedback and appropriate documentation for the user is essential. In complex GO tools, all arguments should be checked at the beginning of the GO tool using the no-op command (has no effect) "QUERY/IGNORE". Thus the GO tool land.jnl might contain these lines at the beginning:

```
! check the argument
QUERY/IGNORE $1"1|red|green|blue|<must be red, green, or blue"
```

Once argument errors have been trapped and reported, the lengthy error text would not be needed again in the GO tool.

GO tools that use arguments should also be carefully documented. There are numerous examples provided with Ferret; try, for example, the Unix commands

```
% Fgo -more fland.jnl
% Fgo -more stick_vectors
```

or

```
% Fgo -more squeeze_colors
```

## Ch1 Sec5.3.5. Flow Control in GO tools

There are several Ferret commands and techniques to assist with flow control in your GO scripts.

**GO (subroutines)**

The GO command may be used inside of a GO script (tool) to execute another (nested) GO script. If an error occurs inside of a nested GO script and SET MODE IGNORE_ERROR has not been issued then the GO script will be interrupted and control returns to the command line.

**REPEAT (looping)**

The REPEAT command may be used to execute loops within Ferret. The loop "counter" may be an index (I,J,K, or L) or a world coordinate (longitude, latitude, depth, or time). The increment between loop iterations need not correspond to the spacing of points on a grid. When used in conjunction with the "d" options of SET REGION, such as SET REGION/DI="-5:-5" the loops may be used to zoom in or out of a region or to pan a limited-width window of view across a larger region. See the Advanced Movie-Making section (p. 124) of this manual for further details.

**IF-THEN-ELSE (conditional execution)**

An IF-THEN-ELSE syntax can be used to conditionally execute Ferret commands. It may be used in two styles—single line and multi-line. See the IF command (p. 267) in the Commands Reference section of this manual for further details.

## Ch1 Sec5.3.6. Debugging GO tools

As the complexity of Ferret GO scripts increases it becomes more challenging to locate and correct errors in GO scripts. This is especially true if, as so many GO scripts do, the scripts are made silent by containing the command CANCEL MODE VERIFY. In a silent script it can be unclear from where within the script an error message is originating.

A special VERIFY mode has been provided to assist with locating the source of these error messages

```
SET MODE VERIFY:ALWAYS
```

The ALWAYS argument to this command instructs Ferret to ignore CANCEL MODE VERIFY commands inside of command files. All of the script commands that Ferret executes will be echoed when this mode is set. Error messages will appear with the commands that generated them. To restore normal non-debugging operations issue CANCEL MODE VERIFY or SET MODE VERIFY (no argument) interactively from the `yes?` prompt.

Complex webs of variable definitions (defined with LET or DEFINE VARIABLE) may also create challenges for debugging scripts. See Debugging Complex Hierarchies of Expressions (p. 100) for further discussion of this topic.

## Ch1 Sec6. SAMPLE DATA SETS

A number of demonstration data sets are included with this distribution. Several of these data sets are used by the demonstration "GO" files, above. The data sets should be accessible simply by typing the Ferret command

```
yes? USE data_set_name
yes? USE coads_climatology
```
for example,

| Data set | Description |
|---|---|
| etopo120 | relief of the earth's surface at 120-minute resolution |
| etopo60 | relief of the earth's surface at 60-minute resolution |
| levitus_climatology | subset of the Climatological Atlas of the World Oceans by Sydney Levitus (**Note:** the updated World Ocean Atlas, 1994, is also available with Ferret) |

| | |
|---|---|
| coads_climatology | 12-month climatology derived from 1946–1989 of the Comprehensive Ocean/Atmosphere Data Set |
| monthly_navy_winds | monthly-averaged Naval Fleet Numerical Oceanography Center global marine winds (1982–1990) |
| esku_heat_budget | Esbensen-Kushnir 4×5 degree monthly climatology of the global ocean heat budget (25 variables) |

## Ch1 Sec7. UNIX TOOLS

A number of tools are provided with Ferret to assist with Unix-level activities: on-line help, converting data to Ferret's formats, locating files, etc. They are located in the Ferret installation area—typically $FER_DIR/bin. See the chapter "Copmuting Environment", section "Setting Up an Account" (p. 185), if the tools are not available on-line. They are described below.

**Faddpath** Usage: Faddpath new_path

Faddpath will add a new path name to the default lists of directories that Ferret searches a) in response to the SET DATA command; b) when looking for grid definition files; c) when looking for data files.

**Fapropos** Usage: Fapropos string (i.e. `% Fapropos regridding`)

Fapropos searches the Ferret User's Guide for all occurrences of the given word or string. The string is not case sensitive. If the string contains multiple words it must be enclosed in quotation marks. Fapropos will list all lines of the User's Guide that contain the word or string and report their line numbers. The line numbers may be used with Fhelp to enter the User's Guide at the desired location.

**Fdata** Usage: Fdata data_file_substring

Searches the list of directories contained in the environment variable FER_DATA to find the data files whose names contain the indicated substring. For example,

```
% Fdata coads
```

locates the data files containing "coads" in their names. (Use this command to locate NetCDF data sets by giving the string "cdf".)

**Fdescr** Usage: Fdescr des_name_substring

Searches the list of directories contained in the environment variable FER_DESCR to find the descriptor files whose names contain the indicated substring. For example,

```
% Fdescr coads
```

locates the descriptor files containing "coads" in their names. ("Fdescr .des" will list all accessible descriptors.)

**Fenv** Usage: Fenv

Prints the values of environment variables used by Ferret

**Fgo** Usage: Fgo name_substring

Searches the list of directories contained in the environment variable FER_GO to find the GO command files whose names contain the indicated substring. For example,

```
% Fgo grid
```

locates the Ferret tools that contain "grid".

**Fgrids** Usage: Fgrids gridfile_substring

Searches the list of directories contained in the environment variable FER_GRIDS to find the grid definition files whose names contain the indicated substring. For example,

```
% Fgrids fnoc
```

locates the grid definition files containing "fnoc" in their names. ("`Fgrids .grd`" will list all accessible grid files.)

**Fhelp** Usage: Fhelp line_number or Fhelp string

Fhelp enters the Ferret User's Guide beginning at the indicated line number or at the first occurrence of the given string. The string, if used, is not case sensitive. The Unix "more" command is used to access the User's Guide. The most commonly used "more" commands are documented under Ftoc.

Examples:
```
% Fhelp  1136
% Fhelp  "modulo axis"
```

**Fman** Usage: Fman

(Not yet implemented.) Enters the Ferret User's Guide as on-line, formatted hypertext.

**Fpalette** Usage: Fpalette name_substring

Searches the list of directories contained in the environment variable FER_PALETTE to find the palette files whose names contain the indicated substring. For example,

```
% Fpalette blue
```

locates the palette files containing "blue" in their names.

**Fpurge** Usage: Fpurge filename_template

Fpurge is a support routine to manage multiple versions of files created by Ferret—particularly journal files and graphic metafiles. Fpurge will remove all versions of a file except the current version. For example, "Fpurge ferret.jnl" will eliminate all past versions of ferret.jnl in the current directory.

**Fsort** Usage: Fsort filename_template

Fsort is a support routine for sorting file versions. Fsort reorders the incorrect ordering of emacs-style version numbers assigned by the Unix "ls" utility. For example, when sorting, ls will place filename.~19~ before filename.~2~. "Fsort filename*" will take care of this problem. Fsort may be used in Unix pipes.

**Ftoc** Usage: Ftoc

Ftoc enters the table of contents of the Ferret User's Guide using the Unix "more" command. Within "more" the following are the most commonly used commands:

| | |
|---|---|
| ? | interactive help for "more" |
| q | exit (quit) |
| space | advance to next screen |
| return | advance to next line |
| b | back one screen |
| /string | locate the next occurrence of "string" (**Note:** the string is case sensitive) |

## Ch1 Sec8. HELP

## Ch1 Sec8.1. Unix on-line help

On Unix systems interactive Ferret help is available from the command line. If multiple windows are not available on your system the ^Z key can be used to suspend the current Ferret session and access the help; the Unix "fg" command resumes the suspended session.

Several Unix commands provide assistance with rapidly locating information in the Ferret User's Guide. The entire Ferret User's Guide is available on-line as document $FER_DIR/doc/ferret_users_guide.txt. A printable version is also available in PostScript: $FER_DIR/doc/ferret_users_guide.ps.

These commands are available to access the Ferret User's Guide:

| | |
|---|---|
| Ftoc | browse the table of contents of the User's Guide |
| Fapropos | locate words or character strings in the User's Guide |
| Fhelp | enter and browse the User's Guide |
| Fman | enter and browse the User's Guide as formatted hypertext (not yet implemented) |

Normally Ftoc or Fapropos is used first to locate the desired information in the User's Guide. Then Fhelp is used to enter the User's Guide at the selected location.

## Ch1 Sec8.2. Examples and demonstrations

As discussed earlier in this chapter (Getting Started, GO files), the demonstrations that come with the Ferret distribution are a source of help. See the introductory chapter, section "Demonstration files," (p. 14) for a list of demonstrations, or look in $FER_DIR/examples; you may find something that addresses your problem.

## Ch1 Sec8.3. Help from within Ferret

Typing "help" while running Ferret will give you information on using the Unix tool Fhelp to access the User's Guide.

The Ferret command SHOW COMMANDS will list all Ferret commands; SHOW COMMAND "command" will display all qualifiers for the specified command.

The Ferret command SHOW FUNCTIONS lists all Ferret functions and their arguemnts. SHOW FUNCTION *string* will show all functions containing the string "string". SHOW FUNCTIONS EXTERNAL shows the names and arguments of external functions (see External Functions Chapter, page 215)

The Ferret command SHOW TRANSFORMS lists all Ferret transforms, including variable transforms and regridding transforms.

If you want to get details on a script, type 'GO/HELP scriptname" to see the documentation at the start of the script. For example:

```
GO/HELP land
```

When writing scripts, include documentation listing the purpose of the script and its arguments in the first few lines of the script. Then this feature will let you and others who may use the script get instant information about it.

## Ch1 Sec8.4. Web-based information

From the Ferret web page, at http://ferret.wrc.noaa.gov/Ferret, see these sections:

1. Ferret support policy outlines the support available to users and sources of information

2. FAQ section discusses many topics where questions often arise.

3. Email archives, which are searchable and contain questions and solutions from the Ferret users group.

4. Documentation section, including release notes, this manual which is updated regularly on the web, and on-line information on demonstration scripts, data formats, and the Plot Plus graphics used by Ferret.

# Chapter 2: DATA SET BASICS

## Ch2 Sec1. OVERVIEW

Ferret accepts input data from both ASCII and binary files and recognizes two standardized, self-describing data formats—NetCDF, and TMAP. Network Common Data Format (NetCDF) is the suggested method of data storage.

SET DATA_SET or just SET DATA specifies a data set for access. ASCII and binary files can be read using SET DATA/EZ (also known as "FILE"). To unambiguously specify the format of a data set, include the extension .cdf or .des in its name, or use the qualifier /FORMAT=CDF.

To examine what each data set consists of (variables, grids, etc.) after specifying them with SET DATA, use SHOW DATA. This command displays the variables in the data set and over what geographical and time ranges they are defined.

Here is an example of Ferret's output:

```
       yes? SET DATA coads_climatology
       yes? SHOW DATA
      currently SET data sets:
     1> /home/e1/tmap/fer_dsets/descr/coads_climatology.des  (default)
name     title                          I         J         K         L
SST      SEA SURFACE TEMPERATURE      1:180     1:90      1:1       1:12
AIRT     AIR TEMPERATURE              1:180     1:90      1:1       1:12
SPEH     SPECIFIC HUMIDITY            1:180     1:90      1:1       1:12
WSPD     WIND SPEED                   1:180     1:90      1:1       1:12
UWND     ZONAL WIND                   1:180     1:90      1:1       1:12
VWND     MERIDIONAL WIND              1:180     1:90      1:1       1:12
SLP      SEA LEVEL PRESSURE           1:180     1:90      1:1       1:12
```

If multiple data sets have been requested in a single Ferret session, the last requested will be the default data set. To specify other data sets, use the name of the data set or the number of the set as given by the SHOW DATA statement. For example:

```
yes? LIST/D=2  temp
```

will list the data for the variable "temp" in data set number 2 as displayed by SHOW DATA/BRIEF, while

```
yes? LIST temp[D=levitus_climatology] - temp[D=coads_climatology]
```

will list the differences between the variable "temp" in data set "levitus_climatology" and data set "coads_climatology."

If a filename begins with a number, Ferret does not recoginze it, but the file may be specified using its unix  pathname, e.g.

```
yes? use "./123"
```

or

```
yes? file/var=a "./45N_180W.dat"
```

## Ch2 Sec2. NETCDF DATA

The Network Common Data Format (NetCDF) is an interface to a library of data access routines for storing and retrieving scientific data. NetCDF allows the creation of data sets which are self-describing and platform-independent. NetCDF was created under contract with the Division of Atmospheric Sciences of the National Scientific Foundation and is available from the Unidata Program Center in Boulder, Colorado (unidata.ucar.edu).

See the chapter "Converting Data to NetCDF" (p. 193), for a complete description of how to create NetCDF data sets or how to convert existing data sets into NetCDF.

To output a variable in NetCDF, simply use:

```
yes?  LIST/FORMAT=CDF variable_name
```

LIST/FORMAT=CDF (alias SAVE) can also be used with abstract variables:

```
yes? SAVE/FILE=example.cdf/I=1:100 sin(I/100)
```

This will create a file named example.cdf.

The current region and data sets determine the variable names in the saved file and the range over which they are saved. Saved data can then be accessed as follows:

```
yes? USE example
```

(USE is an alias for SET DATA/FORMAT=CDF)

If a filename is not specified, Ferret will generate one. (See command SET LIST/FILE in the Commands Reference section, p. 297). An example of converting TMAP-formatted data to NetCDF goes as follows:

```
yes? SET DATA coads_climatology
yes? SAVE/L=1 sst,airt,uwnd,vwnd
```

These commands will save sst, airt, uwnd, and vwnd at the first time step over their entire regions to a NetCDF file named by Ferret.

One advantage to using NetCDF is that users on a different system (i.e., VMS instead of Unix) with different software (i.e., with an analysis tool other than Ferret) can share data easily with-

out substantial conversion work. NetCDF files are self-describing; with a simple command the size, shape and description of all variables, grids and axes can be seen.

With Ferret version 5.1 , the internal functioning of netCDF reads has been changed when "strides" are involved. Suppose that CDFVAR represent a variable from NetCDF file. In version 5.0 and earlier the command PLOT CDFVAR[L=1:1000:10] would have read the entire array of 1000 points from the file; Ferret's internal logic would have subsampled every 10th point from the resulting array in a manner that was consistent for NetCDF variables, ASCII variables, user defined variables, etc. In V5.1 strides applied to netCDF variables are given special treatment -- subsampling is done by the netCDF library. The primary benefit of this is to make network access to remote data sets via DODS more efficient. A remote satellite image of size, say, 1000x1000 points x 8 bit depth (8 megabytes) can efficiently be previewed using

```
SHADE DODS_VAR[i=1:1000:10,j=1:1000:10]
```

If a grid or axis from a netCDF file is used in the definition of a LET-defined variable (e.g. LET my_X = X[g=sst[D=coads_climatology]]) that variable definition will be invalidated when the data set is canceled (CANCEL DATA coads_climtology, in the preceding example). There is a single exception to this behavior: netCDF files such as climtological_axes.cdf, which define grids or axes that are not actually used by any variables. These grids and axes will remain defined even after the data set, itself, has been canceled. They may be deleted with explicit use of CANCEL GRID or CANCEL AXIS.

## Ch2 Sec2.1. Multi-file NetCDF data sets

Ferret supports collections of NetCDF files that are regarded as a single NetCDF data set. Such data sets are referred to as “MC” (multi CDF) data sets. They are particularly useful to manage the outputs of numerical models. MC data sets use a descriptor file, in the style of TMAP-formatted data sets. The data set is referred to inside Ferret by the name of this descriptor file.

A collection of NetCDF files is suitable to form a multi-file data set if

1) The files are connected through their time axis—each file represents one or more time snapshots of the variables it contains.
2) Each file is self-documenting with respect to the time axis of the variables—even if the time axis represents only a single point. (All of the time axes must be identically encoded with respect to units and date of the time origin.)
3) All non-time-dependent variables in the data set must be contained in the first file of the data set (or those variables will not appear in the merged, MC, data set).

A typical MC descriptor file may be found in the chapter “Converting to NetCDF”, in the section “Creating a multi-NetCDF data set.” (p. 212)

## Ch2 Sec2.2. Non-standard NetCDF data sets

As discussed in the Chapter, "Converting Data to NetCDF," (p. 193) Ferret expects netCDF files to adhere to the COARDS conventions (http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html). If the files do not adhere to the COARDS conventions, Ferret will still attempt to access them. Often, the user can use Ferret controls for regridding, reshaping, and otherwise transforming data to recover the intended file contents.

Here are a few common ways in which NetCDF files may deviate from the COARDS standard and how one may cope with those situations in Ferret.

- Files with disordered coordinates

  In the COARDS conventions an axis (a.k.a. "coordinate variable") must have monotonically-increasing coordinate values. If the coordinates are disordered or repeating in a netCDF file, then Ferret will present the coordinates to the user (in SHOW DATA) as a dependent variable, whose name is the axis name, and it will substitute an axis of the index values 1, 2, 3, ... Note that Ferret will apply this same behavior when files have long irregular axis definitions that exceed Ferret's axis memory capacity.

- Files with reverse-ordered axes

  If the coordinates of an axis are monotonically decreasing, instead of increasing, Ferret will transparently reverse both the axis coordinates and the dependent variables that are defined upon that axis. Note that if Ferret writes a reverse-ordered variable to a new netCDF file (with the SAVE command), the coordinates and data in the output file will be in monotonically increasing coordinate order—reversed from the input file.

  If the values of a dependent variable are reversed, but there is no associated coordinate axis then use attach a minus sign to the corresponding axis orientation in the USE/ORDER= qualifier to designate that the variable(s) should be reversed along the corresponding axis. (Feature not yet implemented as of 5/5/99)

- Files with "invalid" variable names

  The COARDS standard specifies that variable names should begin with a letter and be composed of letters, digits, and underscores. In files where the variable names contain other letters, references to those variable names in Ferret must be enclosed in single quotes.

- Files with permuted axis ordering

  The COARDS standard specifies that if any or all of the dimensions of a variable have the interpretations of "date or time" (a.k.a. "T"), "height or depth" (a.k.a. "Z"), "latitude" (a.k.a. "Y"), or "longitude" (a.k.a. "X") then those dimensions should appear in the relative order T, then Z, then Y, then X in the CDL definition corresponding to the file. In files where the axis ordering has been permuted the command qualifiers USE/ORDER= (Command Refer-

ence, p. 290) allow the user to inform Ferret of the correct permutation of coordinates. Note that if Ferret writes a permuted variable to a new netCDF file (with the SAVE command), the coordinates and data in the output file will be in standard X-Y-Z-T ordering (as indicated in the user's /ORDER specification)—permuted from the original file ordering. See the Command Reference (p. 241) for a complete description of the ORDER qualifier.

## Ch2 Sec3. TMAP-FORMATTED DATA

As of Ferret version 2.30, NetCDF is the suggested format for data storage (see the chapter, "Converting to NetCDF," p. 193). This section describing TMAP information is included only for users who already work with data in TMAP format.

To access TMAP-formatted data sets use

```
SET DATA_SET TMAP_set1, TMAP_set2, ...
```

TMAP_setn must be the name of a descriptor file for a data set that is in TMAP "GT" (grids-at-timesteps) or "TS" (time series) format. ("Ferret" format and "TMAP" format are synonyms.)

If the directory portion of the filename is omitted the environment variable FER_DESCR will be used to provide a list of directories to search. The order of directories in FER_DESCR determines the order of directory searches. If the extension is omitted a default of ".des" will be assumed (if the filename has more than one period, the extension must be given explicitly).

**Descriptors**

For every TMAP-formatted data set there is a descriptor file containing summary information about the contents of the data set. This includes variable names, units, grids, and coordinates. When the command SET DATA_SET is given to Ferret pointing to a GT-formatted or TS-formatted data set, it is the name of the descriptor file that must be specified.

## Ch2 Sec4. BINARY DATA

Ferret can read binary data files that are formatted with and without FORTRAN record length headers (binary files without FORTRAN record length formatting are also known as "stream" files).

## Ch2 Sec4.1. FORTRAN-structured binary files

Files containing record length information are created by FORTRAN programs using the ACCESS="SEQUENTIAL" (the FORTRAN default) mode of file creation and also by Ferret

using LIST/FORMAT=unf. Files that contain FORTRAN record length headers must have all data aligned on a 4-byte boundary. Suppose "rrrr" represents 4 bytes of record length information and "dddd" represents a 4-byte data value. Then FORTRAN-structured files are organized in one of the following two ways:

## Ch2 Sec4.2. Records of uniform length

A FORTRAN-structured file with records of uniform length (3 single-precision floating point data values per record in this figure) looks like this:

```
rrrr dddd dddd dddd rrrr ...
```

FORTRAN code that creates a data file of this type might look something like this (sequential access is the default and need not be specified in the OPEN statement):

```
REAL VARI(10), VAR2(10), VAR3(10)
...
OPEN(UNIT=20,FORMAT='UNFORMATTED',ACCESS='SEQUENTIAL',FILE='MYFILE.DAT')
...
DO 10 I=1,10
WRITE (20) VAR1(I), VAR2(I), VAR3(I)
10 CONTINUE
....
```

To access data from this file, use

```
yes? SET DATA/EZ/FORMAT=UNF/VAR=var1,var2,var3/COL=3  myfile.dat    or,
yes? FILE/FORMAT=UNF/VAR=var1,var2,var3/COLUMNS=3  myfile.dat
```

This is very similar to accessing ASCII data with the addition of the /FORMAT=unf qualifier. The /COLUMNS= qualifier tells Ferret the number of data values per record. Although optional in the above example, this qualifier is required if the number of data values per record is greater than the number of variables being read (examples follow in section "ASCII Data").

## Ch2 Sec4.3. Records of non-uniform length

A FORTRAN-structured file with variable-length records might look like this:

```
rrrr dddd dddd rrrr
rrrr dddd rrrr
rrrr dddd dddd dddd dddd rrrr
etc.
```

With care, it is possible to read a data file containing variable-length records which was created using the simplest unformatted FORTRAN OPEN statement and a single WRITE statement for each variable. Use /FORMAT=stream to read such files. Note that sequential access is the FORTRAN default and does not need to be specified in the OPEN statement:

```
REAL VAR1(1000), VAR2(500)
...
OPEN (UNIT=20, FORMAT="UNFORMATTED", FILE="MYFILE.DAT")
...
WRITE (20) VAR1
WRITE (20) VAR2
....
```

Use the qualifier /SKIP to skip past the record length information (/SKIP arguments are in units of words), and define a grid which will not read past the data values. The /COLUMNS= qualifier can be used when reading multiple variables to specify the number of words separating the start of each variable:

```
yes? DEFINE AXIS/X=1:500:1  xaxis
yes? DEFINE GRID/X=XAXIS  mygrid
yes? FILE/FORMAT=stream/SKIP=1003/GRID=mygrid/VAR=var2  myfile.dat
```

The argument 1003 is the sum of the 1000 data words in record 1, plus 2 words of record length information surrounding the data values in record 1 (variable var1), plus 1 word of record information preceding the data in record 2.

## Ch2 Sec4.4. Stream binary files

Files without embedded record length information are created by FORTRAN programs using ACCESS="DIRECT" in OPEN statements and by C programs using the C studio library. These files can contain a mix of integer and real numbers. The following types can be read from an unstructured file:

| FORTRAN | C | Size in bytes |
|---|---|---|
| INTEGER*1 | char | 1 |
| INTEGER*2 | short | 2 |
| INTEGER*4 | int | 4 |
| REAL*4 | float | 4 |
| REAL*8 | double | 8 |

## Ch2 Sec4.4.1. Simple stream files

Suppose “dddd” represents a 4-byte data value. Then a stream (or “direct access”) binary file of FORTRAN REAL*4 or C floats is:

```
dddd dddd dddd dddd dddd dddd ...
```

The structure of the records is implied by the program accessing the data. FORTRAN code which generates a direct access binary file might look like this:

```
REAL*4 MYVAR(10,5)
...
C Use RECL=40 for machines that specify in bytes

OPEN(UNIT=20, FILE="myfile.dat", ACCESS="DIRECT", RECL=10)
...
DO 100 j = 1, 5
100   WRITE (20,REC=j) (MYVAR(i,j),i=1,10)
....
```

Use the following Ferret commands to read variable “myvar” from this file:

```
yes? DEFINE AXIS/X=1:10:1 x10
yes? DEFINE AXIS/Y=1:5:1 y5
yes? DEFINE GRID/X=x10/Y=y5 g10x5
yes? FILE/VAR=MYVAR/GRID=g10x5/FORMAT=stream  myfile.dat
```

If the file consisted of a set of FORTRAN REAL*8 or C doubles, then the data would look like:

```
dddddddd dddddddd dddddddd ...
```

and the following Ferret commands would read the data into “myvar”:

```
yes? DEFINE AXIS/X=1:10:1 x10
yes? DEFINE AXIS/Y=1:5:1 y5
yes? DEFINE GRID/X=x10/Y=y5 g10x5
yes? FILE/VAR=MYVAR/GRID=g10x5/FORMAT=stream/type=r8  myfile.dat
```

Note the addition of the “type” qualifier. See the FILE command (p. 265) for more details.

Since Ferret represents all variables as REAL*4, some precision is lost when reading in REAL*8 or INTEGER*4 values. Also, some REAL*8 numbers cannot be represented as REAL*4 numbers; the internal Ferret value of such a number is system dependent.

## Ch2 Sec4.4.2. Mixed stream files

Ferret can read binary files that contain a mix of numbers of different type. However, a given Ferret variable can only be one type. Say you have a file containing a mix of REAL*8 and REAL*4 numbers:

```
dddddddd dddd dddddddd dddd dddddddd ...
```

The following would successfully read the file:

```
yes? FILE/VAR=MYDOUBLE,MYFLOAT/GRID=somegrid/FORMAT=stream/type=r8,r4
myfile.dat
```

while:

```
yes? FILE/VAR=MYDOUBLE/GRID=someothergrid/FORMAT=stream/type=r8,r4
myfile.dat
```

would fail.

Stream files with byte-swapped numbers can be read with the /swap qualifier; the /order, and /skip qualifiers are also available (see chapter "Data Set Basics", section "Reading ASCII files," p. 37, for more details on /order and /skip).

## Ch2 Sec5. ASCII DATA

To access ASCII data file sets use

```
yes? SET DATA/EZ   ASCII_file_name    or equivalently
yes? FILE   ASCII_file_name
```

The following are qualifiers to SET DATA/EZ or FILE:

| Qualifier | Description |
|---|---|
| /VARIABLES | names the variables in the file |
| /TITLE | associates a title with the data set |
| /GRID | indicates multi-dimensional data and units |
| /COLUMNS | tells how many data values are in each record |
| /FORMAT | specifies the format of the file |
| /SKIP | skips initial records of the file |
| /ORDER | specifies order of axes (which varies fastest) |

Use command SET VARIABLE to individually customize the variables.

## Ch2 Sec5.1. Reading ASCII files

Below are several examples of reading ASCII data properly. (Uniform record length, FORTRAN-structured binary data are read similarly with the addition of the qualifier /FORMAT= "unf". Seethe chapter on "Data Set Basics", section "Binary Data," p. 33, for other binary types). First, we look briefly at the relationship between Ferret and standard matrix notation.

Linear algebra uses established conventions in matrix notation. In a matrix A(i,j), the first index denotes a (horizontal) row and the second denotes a (vertical) column.

| | | | | | |
|---|---|---|---|---|---|
| A11 | A12 | A13 | ... | A1n | |
| A21 | A22 | A23 | ... | A2n | Matrix A(i,j) |
| ... | | | | | |
| Am1 | Am2 | Am3 | ... | Amn | |

X-Y graphs follow established conventions as well, which are that X is the horizontal axis (and in a geographical context, the longitude axis) and increases to the right, and Y is the vertical axis (latitude) and increases upward (Ferret provides the /DEPTH qualifier to explicitly designate axes where the vertical axis convention is reversed).

In Ferret, the first index of a matrix, i, is associated with the first index of an (x,y) pair, x. Likewise, j corresponds to y. Element Am2, for example, corresponds graphically to x=m and y=2.

By default, Ferret stores data in the same manner as FORTRAN—the first index varies fastest. Use the qualifier /ORDER to alter this behavior. The following examples demonstrate how Ferret handles matrices.

**Example 1—1 variable, 1 dimension**

1a) Consider a data set containing the height of a plant at regular time intervals, listed in a single column:

```
2.3
3.1
4.5
5.6
. . .
```

To access, name, and plot this variable properly, use the commands

```
yes? FILE/VAR=height plant.dat
yes? PLOT height
```

1b) Now consider the same data, except listed in four columns:

```
2.3   3.1   4.5   5.6
5.7   5.9   6.1   7.2
. . .
```

Because there are more values per record (4) than variables (1), use:

```
yes? FILE/VAR=height/COLUMNS=4 plant4.dat
yes? PLOT height
```

**Example 2—2 variables, 1 dimension**

2a) Consider a data set containing the height of a plant and the amount of water given to the plant, measured at regular time intervals:

```
2.3 20.4
3.1 31.2
4.5 15.7
5.6 17.3
. . .
```

To read and plot this data use

```
yes? FILE/VAR="height,water" plant_wat.dat
yes? PLOT height,water
```

2b) The number of columns need be specified only if the number of columns exceeds the number of variables. If the data are in six columns

```
2.3 20.4   3.1 31.2   4.5 15.7
5.6 17.3 ...
```

use

```
yes? FILE/VAR="height,water"/COLUMNS=6 plant_wat6.dat
yes? PLOT height,water
```

**Example 3—1 variable, 2 dimensions**

3a) Consider a different situation: a greenhouse with three rows of four plants and a file with a single column of data representing the height of each plant at a single time (successive values represent plants in a row of the greenhouse):

```
3.1
2.6
5.4
4.6
3.5
6.1
. . .
```

If we want to produce a contour plot of height as a function of position in the greenhouse, axes will have to be defined:

```
yes? DEFINE AXIS/X=1:4:1 xplants
yes? DEFINE AXIS/Y=1:3:1 yplants
yes? DEFINE GRID/X=xplants/Y=yplants gplants
yes? FILE/VAR=height/GRID=gplants greenhouse_plants.dat
yes? CONTOUR height
```

When reading data the first index, x, varies fastest. Schematically, the data will be assigned as follows:

```
        x=1        x=2         x=3        x=4
y=1     3.1        2.6         5.4        4.6
y=2     3.5        6.1 . . .
y=3    . . .
```

3b) If the file in the above example has, instead, 4 values per record:

```
3.1   2.6   5.4   4.6
3.5   6.1  . . .
```

then add /COLUMNS=4 to the FILE command:

```
yes? FILE/VAR=height/COLUMNS=4/GRID=gplants greenhouse_plants.dat
```

### Example 4—2 variables, 2 dimensions

Like Example 3, consider a greenhouse with three rows of four plants each and a data set with the height of each plant and the length of its longest leaf:

```
3.1     0.54
2.6     0.37
5.4     0.66
4.6     0.71
3.5     0.14
6.1     0.95
.        .
.        .
```

Again, axes and a grid must be defined:

```
yes? DEFINE AXIS/X=1:4:1 xht_leaf
yes? DEFINE AXIS/Y=1:3:1 Yht_leaf
yes? DEFINE GRID/X=xht_leaf/Y=yht_leaf ght_leaf
yes? FILE/VAR="height,leaf"/GRID=ght_leaf greenhouse_ht_lf.dat
yes? SHADE height
yes? CONTOUR/OVER leaf
```

The above commands create a color-shaded plot of height in the greenhouse, and overlay a contour plot of leaf length. Schematically, the data will be assigned as follows:

```
          x=1          x=2          x=3          x=4
       ht ,  lf     ht ,  lf
y=1    3.1, 0.54    2.6, 0.37    5.4, 0.66    4.6, 0.71
y=2    3.5, 0.14    6.1, 0.95 . . .
y=3    . . .
```

### Example 5—2 variables, 3 dimensions (time series)

Consider the same greenhouse with height and leaf length data taken at twelve different times. The following commands will create a three-dimensional grid and a plot of the height and leaf length versus time for a specific plant.

```
yes? DEFINE AXIS/X=1:4:1 xplnt_tm
yes? DEFINE AXIS/Y=1:3:1 yplnt_tm
yes? DEFINE AXIS/T=1:12:1 tplnt_tm
yes? DEFINE GRID/X=xplnt_tm/Y=yplnt_tm/T=tplnt_tm gplant2
yes? FILE/VAR="height,leaf"/GRID=gplant2 green_time.dat
yes? PLOT/X=3/Y=2 height, leaf
```

### Example 6—1 variable, 3 dimensions, permuted order (vertical profile)

Consider a collection of oceanographic measurements made to a depth of 1000 meters. Suppose that the data file contains only a single variable, salt. Each record contains a vertical profile (11 values) of a particular x,y (long,lat) position. Supposing that successive records are successive longitudes, the data file would look as follows (assume the equivalencies are not in the file):

```
            z=0   z=10  z=20  . . .
x=30W,y=5S  35.89 35.90 35.93 35.97 36.02 36.05 35.96 35.40 35.13 34.89 34.72
x=29W,y=5S  35.89 35.91 35.94 35.97 36.01 36.04 35.94 35.39 35.13 34.90 34.72
          . . .
```

Use the qualifier /DEPTH= when defining the Z axis to indicate positive downward, and /ORDER when setting the data set to properly read in the permuted data:

```
yes? DEFINE AXIS/X=30W:25W:1/UNIT=degrees salx
yes? DEFINE AXIS/Y=5S:5N:1/UNIT=degrees saly
yes? DEFINE AXIS/Z=0:1000:100/UNIT=meters/DEPTH salz
yes? DEFINE GRID/X=salx/Y=saly/Z=salz salgrid
yes? FILE/ORDER=zxy/GRID=salgrid/VAR=sal/COL=11 sal.dat
```

## Ch2 Sec6. TRICKS TO READING BINARY AND ASCII FILES

Since binary and ASCII files are found in a bewildering variety of non-standardized formats a few tricks may help with reading difficult cases.

- Sometimes variables are interleaved with data axes in unstructured (stream) binary files. A simple trick is to read them all as a single variable, say, “Vall,” in which the sequence of variables in the file V1, V2, V3, ... is regarded as an axis of the grid. Then extract the variables by defining V1 = Vall[I=1] (if the I axis was used, else J=1, K=1, or L=1) as needed.

- In some ASCII files the variables are presented as blocks—a full grid of variable 1, then a full grid of variable 2, etc. These files may be read using Unix soft links so that the same file can be opened as several Ferret data sets. Then use the FILE command to point separately to each soft link using the /SKIP qualifier to locate the correct starting point in the file for each variable. For example,

  Unix commands:

  ```
  ln -s my_data my_dat.v1
  ln -s my_data my_dat.v2
  ln -s my_data my_dat.v3
  ```

  Ferret commands:

  ```
  yes? FILE/SKIP=0/VAR=v1 my_dat.v1
  yes? FILE/SKIP=100/VAR=v2 my_dat.v2
  yes? FILE/SKIP=200/VAR=v3 my_dat.v3
  ```

- If an ASCII file contains a repeating sequence of records try describing the entire sequence using a single FORTRAN FORMAT statement. An example of such a statement would be (3F8.4,2(/5F6.2)). The slash character and the nested parentheses allow multi-record groups to appear as a single format. Note that the /COLUMNS qualifier should reflect the total number of columns in the repeating group of records.

- If an ASCII or binary file contains gridded data in which the order of axes is not X-Y-Z-T read the data in (which results in the wrong axis ordering) and use the LIST/ORDER= to permute the order on output. The resulting file will have the desired axis ordering.

- If the times and geographical coordinate locations of the grid are inter-mixed with the dependent variables in the file then 1) issue a FILE command to read the coordinates only; 2) use DEFINE AXIS/FROM_DATA to define axes and DEFINE GRID to define the grid; 3) use FILE/GRID=mygrid to read the file again.

## Ch2 Sec7. ACCESS TO REMOTE DATA SETS WITH DODS

- **What is DODS?**

  DODS, the Distributed Oceanographic Data System, allows users to access data anywhere from the internet using a variety of client/server methods, including Ferret. Employing technology similar to that used by the World Wide Web, DODS and Ferret create a powerful tool for the retrieval, sampling, analyzing and displaying of datasets; regardless of size or data format (though there are data format limitations).

  For more information on DODS, please see the DODS home page at

  http://unidata.ucar.edu/packages/dods/

  Similar to the WWW, DODS is an emerging technology and is under development. As a result, it is likely that the details with which things are accomplished will be changing.

- **Accessing Remote Data Sets**

  Datasets are accessed through Ferret using their raw Universal Resource Locator (URL) address. For example, to access the Coads climatology, hosted at PMEL:

```
yes? use
"http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc"
```

Once the dataset has been initialized, it is used just like any other local dataset.

```
yes? list/x=140w/y=2n/t="16-Feb" sst
            SEA SURFACE TEMPERATURE (Deg C)
            LONGITUDE: 141W
            LATITUDE: 1N
            TIME: 15-FEB 16:29
            DATA SET:
http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc
         26.39
```

We have developed some general scripts (available at http://ferret.wrc.noaa.gov/Ferret/Dods/) which will assist in both finding and using data with Ferret and DODS. This script illustrates the use of datasets which are located on our server at PMEL.

```
yes? go dods
_____________________
DODS data suppliers currently known to your Ferret installation:
* * * * * * * * in .
dods_cdc.jnl: "Use" a dods data set from NOAA/CDC
dods_fsu.jnl: "Use" a dods data set from FSU
dods_jpl.jnl: "Use" a dods data set from JPL
dods_pmel.jnl: "Use" a dods data set from NOAA/PMEL
dods_uri.jnl: "Use" a dods data set from URI/GSO
```

***The techniques for cataloging and organizing distributed DODS data are still under development. The Ferret scripts to assist with DODS access are subject to change.***

```
yes? go dods_pmel
_________________

The available data sets are:
   coads_climatology.nc levitus_climatology.nc

The base URL is: http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/

yes? go dods_pmel coads_climatology.nc
yes? sh data/br
currently SET data sets:
1> http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc
(default)
yes? list/x=140w/y=2n/t="16-Apr" sst
       SEA SURFACE TEMPERATURE (Deg C)
       LONGITUDE: 141W
       LATITUDE: 1N
       TIME: 16-APR 13:27
       DATA SET:
http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc
            27.07
```

- **Debugging Access to Remote DODS Data Sets**

To find out more information about a particular dataset, or to debug problems, there are three elements of the dataset which may be accessed via a web browser. To access this information, merely append a dds, das, or info to the dataset name. For example:

```
http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc.dds
```

DDS stands for Data Description Structure and this will return a text description of the data sets structure.

```
http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc.das
```

DAS stands for Dataset Attribute Structure and this will return a text description of attributes assigned to the variables in the data set.

```
http://ferret.wrc.noaa.gov/cgi-bin/nph-nc/data/coads_climatology.nc.inf
o
```

This will return a text description of the variables in the dataset.

- **Sharing Data Sets via DODS**

  One of the most powerful aspect of DODS is the ease with which it allows for the sharing of data. With just a few simple steps, anyone running a web server can also be a DODS data server, thereby allowing data set access to anyone with an internet connection.

  Simply copying a few precompiled binaries into the cgi-bin directory of an already configure httpd server is all it takes to become a DODS server. Once the server is configured, adding or removing data sets is as simple as copying them to the server data directory or deleting them from that directory.

  This ability has such immense potential that it bears extra emphasis. Imagine that within seconds of finishing a model run, a remote colleague is able to look at your dataset with whatever DODS client he/she desires, be it Ferret, or Matlab, etc. No need for you to package up the data or for your colleague to download and/or reformat it, it is ready to be analyzed right away.

- **DODS caching**

This feature allows caching of frequently accessed DODS-served datasets to produce a quicker response when requesting the remote data.

The first time you access a dods data set, a file in the users home directory will be created called dodsrc and will contain:

```
USE_CACHE=1     ! Turn cache on or off - 0=off, 1=on
MAX_CACHE_SIZE=20   ! max cache size in Mbytes
MAX_CACHED_OBJ=5    ! max cache entry size in Mbytes
IGNORE_EXPIRES=0    ! 0: Honor expiration information from servers,1: ignore them
CACHE_ROOT=/home/myname/.dods_cache/  !pathname to cache directory
DEFAULT_EXPIRES=86400    ! expiration time, in seconds, of cached data
```

Also created will be a .dods_cache directory, which by default (as mentioned above) is created in the users home directory. This is where all the cached information is stored. To clear the DODS cache, simply delete the .dods_cache directory and all of it's contents (for example, rm -r ~/.dods_cache). This directory will be recreated and repopulated with caching information the next time data is accessed via DODS and caching is turned on. Of course, all of the above values can be modified to better suit individual needs, and will be incorporated the next time Ferret is run. For example, to turn caching off, simply set USE_CACHE to 0, and restart Ferret.

For more detailed information on setting up a DODS server, please see the DODS home page (http://unidata.ucar.edu/packages/dods).

# Chapter 3: VARIABLES AND EXPRESSIONS

## Ch3 Sec1. VARIABLES

Variables are of 2 kinds:

1) file variables (read from disk files)
2) user-defined variables (defined by the user with LET command)

Both types may be accessed identically in all commands and expressions.

Variables, regardless of kind, possess the following associated information:

1) grid—the underlying coordinate structure
2) units
3) title
4) title modifier (additional explanation of variable)
5) flag value for missing data points

Use the commands SHOW DATA and SHOW VARIABLES to examine file variables and user-defined variables, respectively.

The pseudo-variables I, J, K, L, X, Y, Z, T and others may be used to refer to the underlying grid locations and characteristics and to create abstract variables.

## Ch3 Sec1.1. Variable syntax

Variables in Ferret are referred to by names with optional qualifying information appended in square brackets. See DEFINE VARIABLE (p. 260) for a discussion of legal variable names.

The information that may be included in the square brackets includes

```
D=data_set_name_or_number       ! indicate the data set
G=grid_or_variable_name         ! request a regridding
X=,Y=,Z=,T=,I=,J=,K=,L=         ! specify region and transformation
```

See the chapter “Grids and Regions”, section “Regions” (p. 116) for more discussion of the syntax of region qualifiers and transformations.

Some examples of valid variable syntax are

```
Myvar               ! data set and region as per current context
myvar[D=2]          ! myvar from data set number 2 (see SHOW DATA, p.
318)
myvar[D=a_dset]     ! myvar from data set a_dset.cdf or a_dset.des
```

```
myvar[D=myfile.txt] ! myvar from file myfile.txt
myvar[G=gridname]   ! myvar regridded to grid gridname
myvar[G=var2]       ! myvar regridded to the grid of var2
                     ! which is in the same data set as myvar
myvar[G=var2[D=2]]  ! myvar regridded to the grid of var2
                     ! which is in data set number 2
myvar[GX=axisname]  ! myvar regridded to a dynamic grid which
                     ! has X axis axisname
myvar[GX=var2]      ! myvar regridded to a dynamic grid which
                     ! has the X axis of variable var2
myvar[I=1:31:5]     ! myvar subsampled at every 5th point
                     ! (regridded to a subsampled axis)
myvar[X=20E:50E:5]  ! myvar subsampled at every 5 degrees
                     ! (regridded to a 5-deg axis by linear
interpolation)
```

## Ch3 Sec1.2. File variables

File variables are stored in disk files. Input data files can be ASCII, binary, NetCDF, or TMAP-formatted (see the chapter "Data Set Basics", p. 29). File variables are made available with the SET DATA (alias USE) command.

In some netCDF files the variable names are not consistent with Ferret's rules for variable naming. They may be case-sensitive (for example, variables "v" and "V" defined in the same file), may be restricted names such as the Ferret pseudo-variable names I, J, K, L, X, Y, Z, T, XBOX, YBOX, ZBOX, or TBOX, or them may include "illegal" characters such as "+", "-", "%", blanks, etc. To access such variable names in Ferret file simply enclose the name in single quotes. For example,

```
yes? PLOT 'x'
yes? CONTOUR 'SST from MP/RF measurements'
```

## Ch3 Sec1.3. Pseudo-variables

Pseudo-variables are variables whose values are coordinates or coordinate information from a grid. Valid pseudo-variables are

| | | |
|---|---|---|
| X – x axis coordinates | I – x axis subscripts | XBOX – size of x grid box |
| Y – y axis coordinates | J – y axis subscripts | YBOX – size of y grid box |
| Z – z axis coordinates | K – z axis subscripts | ZBOX – size of z grid box |
| T – t axis coordinates | L – t axis subscripts | TBOX – size of t grid box |

A grid box is a concept needed for some transformations along an axis; it is the length along an axis that belongs to a single grid point and functions as a weighting factor during integrations and averaging transformations.

The pseudo-variables I, J, K, and L are subscripts; that is, they are a coordinate system for referring to grid locations in which the points along an axis are regarded as integers from 1 to the number of points on the axis. This is clear if you look at one of the sample data sets:

```
yes? USE levitus_climatology
yes? SHOW DATA
    1> /home/e1/tmap/fer_dsets/descr/levitus_climatology.des  (default)
       Levitus annual climatology (1x1 degree)
             diagnostic variables: NOT available
      name   title                    I         J         K       L
      TEMP   TEMPERATURE            1:360     1:180     1:20     ...
        ... on grid GLEVITR1   X=20E:20E(380) Y=90S:90N  Z=0m:5000m
      SALT   SALINITY               1:360     1:180     1:20     ...
        ... on grid GLEVITR1   X=20E:20E(380) Y=90S:90N  Z=0m:5000m
```

We see that there are 20 points along the z-axis (1:20 under K), for example, and that the z-axis coordinate values range from 0 meters to 5000 meters. Pseudo-variables depend only on the underlying grid, and not on the variables (in this case, temperature and salt).

**Examples: Pseudo-variables**

1) `yes? LIST/I=1:10 I`
2) `yes? LET xflux = u * vbox[G=u]`

## Ch3 Sec1.3.1. Grids and axes of pseudo-variables

The name of a pseudo-variable, alone, (”I”, “T”, “ZBOX”, etc.) is not sufficient to determine the underlying axis of the pseudo-variable. The underlying axis may be specified explicitly, may be inherited from other variables used in the same expression, may be generated dynamically, or may be inherited from the current default grid. The following examples illustrate the possibilities:

```
TEMP + Y         ! pseudo-variable Y inherits the y axis of variable
TEMP
Y[G=TEMP]        ! explicit: Y refers to the y axis of variable TEMP
Y[GY=axis_name] ! explicit: Y refers to axis axis_name
Y[Y=0:90:2]      ! y axis is dynamically generated (See “dynamic axes”>,
                 ! p. 102)
```

In the expression

```
LET A = X + Y
```

in which the definition provides no explicit coaching, nor are there other variables from which Y can inherit an axis, the axis of Y will be inherited from the current default grid. The current default grid is specified by the SET GRID command and may be queried at any time with the SHOW GRID command. SHOW GRID will respond with “Default grid for DEFINE VARIABLE is grid”.

Note that when pseudo-variables are buried within a user variable definition they do not inherit from variables used in conjunction with the user variable. For example, contrast these expressions involving pseudo-variable Y

```
USE coads_climatology ! has variable SST
LET A = Y             ! Y buried inside variable A (axis indeterminate)
LIST SST + A          ! y axis inherited from current default grid
LIST SST + Y          ! y axis inherited from grid of SST
LIST SST + A[G=SST]   ! y axis inherited from grid of SST
```

## Ch3 Sec1.4. User-defined variables

New variables can be defined from existing variables and from abstract mathematical quantities (such as COS(latitude)) with command DEFINE VARIABLE (alias LET). The section later in this chapter, Defining New Variable (p. 99) expands on this capability.

See command DEFINE VARIABLE (p. 260) and command LET (p. 270) in the Commands Reference.

**Examples: User-defined variables**

1) `yes? LET/TITLE="Surface Relief x1000 (meters)" r1000=rose/1000`
2) `yes? LET/TITLE="Temperature Deviation" tdev=temp - temp[Z=@ave]`

## Ch3 Sec1.5. Abstract variables

Ferret can be used to manipulate abstract mathematical quantities such as SIN(x) or EXP(k*t)—quantities that are independent of file variable values. Such quantities are referred to as abstract expressions.

**Example: Abstract variables**

Contour the function
`COS(a*Y)/EXP(b*T)  where a=0.25 and b=-0.02`

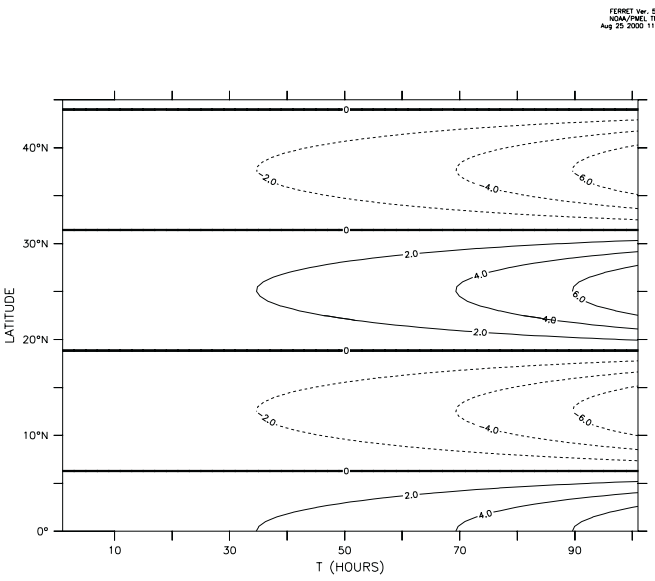
over the range
`Y=0:45 (degrees) and  T=1:100 (hours)`

with a resolution of
0.5 degree on the Y axis and 2 hours on the T axis.

Quick and dirty solution:

```
yes? CONTOUR COS(0.25*Y[Y=0:45:0.5])/EXP(-0.2*T[T=1:100:2])
```

Nicer (Figure 3_1); plot is documented with correct units and titles):




COS(A*Y)/EXP(B*T)

Figure 3_1

```
yes? DEFINE AXIS/Y=0:45:0.5       /UNIT=DEGREES yax
yes? DEFINE AXIS/T=1:100:2        /UNIT=HOURS tax
yes? DEFINE GRID/T=tax/Y=yax my_grid
yes? SET GRID my_grid
yes? LET a=0.25
yes? LET b=-0.02
yes? CONTOUR COS(a*Y)/EXP(b*T)
```

See the chapter “Grids and Regions”, section “Grids” (p. 101), for more information on grids.

## Ch3 Sec1.6. Missing value flags

Data values that are absent or undefined for mathematical reasons (e.g., 1/0) will be represented in Ferret with a missing value flag. In SHADE outputs a missing value flag embedded at some point in a variable will result in a transparent rectangular hole equal to the size of the grid cell of the missing value. In a CONTOUR or FILL plot it will result in a larger hole—extending past the grid box edge to the coordinate location of the next adjacent non-missing point—since contour lines cannot be interpolated between a missing value and its neighboring points. In the output of the LIST command for cases where the /FORMAT qualifier is not used the missing value will be represented by 4 dots (“....”). For cases where LIST/FORMAT=FORTRAN-format is used the numerical value of the missing value flag will be printed using the format provided.

## Ch3 Sec1.6.1. Missing values in input files

Ferret does not impose a standard for missing value flags in input data sets; each variable in each data set may have its own distinct missing value flag(s). The flag(s) actually in use by a data set may be viewed with the SHOW DATA/VARIABLES command. If no missing value flag is specified for a data set Ferret will assume a default value of –1.E+34.

For EZ input data sets, either binary or ASCII, the missing data flag may be specified with the SET VARIABLE/BAD= command. A different value may be specified for each variable in the data set.

For NetCDF input data sets the missing value flag(s) is indicated by the values of the attributes "missing_value" and "_FillValue." If both attributes are defined to have different values both will be recognized and used by Ferret as missing value indicators, however the occurrences of _FillValue will be replaced with the value of missing_value as the data are read into Ferret's memory cache so that only a single missing value flag is apparent inside of Ferret. The command SET VARIABLE/BAD= can also be applied to NetCDF variables, thereby temporarily setting a user-imposed value for _FillValue.

## Ch3 Sec1.6.2. Missing values in user-defined variables

User-defined variables may in general be defined as expressions involving multiple variables. The component variables need not in general agree in their choice of missing value flags. The result variable will inherit the bad value flag of the first variable in the expression. If the first component in the expression is a constant or a pseudo-variable, then Ferret imposes its default missing value flag of –1.E+34.

The function MISSING(variable,replacement) provides a limited control over the choice of missing values in user-defined variables. Note, however, that while the MISSING function will replace the missing values with other values it will not change the missing value flag. In other words, the replacement values will no longer be regarded as missing.

## Ch3 Sec1.6.3. Missing values in output NetCDF files

Values flagged as missing inside Ferret will be faithfully transferred to output files—no substitution will occur as the data are written. In the case of NetCDF output files both of the attributes missing_value, and _FillValue will be set equal to the missing value flag.

Under some circumstances it is desirable to save a user-defined variable in a NetCDF file and then to redefine that variable and to append further output. (An example of this is the process of consolidating several files of input, say, moored measurements, into a gridded output.) The process of appending will not change any of the NetCDF attributes—neither long_name (title), units, nor missing_value or _FillValue. If the subsequent variable definitions do not agree in

their choice of missing value flags the resulting output may contain multiple missing value flags that will not be properly documented.

An easy "trick" that avoids this situation is to begin all of the variable definitions with an addition of zero, "LET var = 0 + ...." The addition of zero will not affect the value of the output but it will guarantee that a missing value flag of –1.E+34 will be consistently used. Of course, you will want to use the SET VARIABLE/TITLE= command in conjunction with this approach.

## Ch3 Sec1.6.4. Displaying the missing value flag

If the LIST command is used, missing values are, by default, displayed as "...." To examine the flag as a numerical value, use LIST/FORMAT=(E) (or some other suitable format).

# Ch3 Sec2. EXPRESSIONS

Throughout this manual, Ferret commands that require and manipulate data are informally called "action" commands. These commands are:

PLOT
CONTOUR
FILL (alias for CONTOUR/FILL)
SHADE
VECTOR
POLYGON
WIRE
LIST
STAT
LOAD

Action commands may use any valid algebraic expression involving constants, operators (+,–,*,...), functions (SIN, MIN, INT,...), pseudo-variables (X, TBOX, ...) and other variables.

A variable name may optionally be followed by square brackets containing region, transformation, data set, and regridding qualifiers. For example, "temp", "salt[D=2]", "u[G=temp"], "u[Z=0:200@AVE]".

The expressions may also contain a syntax of:

IF condition THEN expression_1 ELSE expression_2

**Examples: Expressions**

i) `temp ^ 2`
temperature squared

ii) `temp - temp[Z=@AVE]`
for the range of Z in the current context, the temperature deviations from the vertical average

iii) `COS(Y)`
the cosine of the Y coordinate of the underlying grid (by default, the y-axis is implied by the other variables in the expression)

iv) `IF (vwnd GT vwnd[D=monthly_navy_winds]) THEN vwnd ELSE 0`
use the meridional velocity from the current data set wherever it exceeds the value in data set monthly_navy_winds, zero elsewhere.

## Ch3 Sec2.1. Operators

Valid operators are

+
–
*
/
^ (exponentiate)
AND
OR
GT
GE
LT
LE
EQ
NE

## Ch3 Sec2.2. Multi-dimensional expressions

Operators and functions (discussed in the next section, Functions) may combine variables of like dimensions or differing dimensions.

If the variables are of like dimension then the result of the combination is of the same dimensionality as inputs. For example, suppose there are two time series that have data on the same time axis; the result of a combination will be a time series on the same time axis.

If the variables are of unlike dimensionality, then the following rules apply:

1) To combine variables together in an expression they must be "conformable" along each axis.
2) Two variables are conformable along an axis if the number of points along the axis is the same, or if one of the variables has only a single point along the axis (or, equivalently, is normal to the axis).
3) When a variable of size 1 (a single point) is combined with a variable of larger size, the variable of size 1 is "promoted" by replicating its value to the size of the other variable.
4) If variables are the same size but have different coordinates, they are conformable, but Ferret will issue a message that the coordinates on the axis are ambiguous. The result of the combination inherits the coordinates of the FIRST variable encountered that has more than a single point on the axis.

**Examples:**

Assume a region J=50/K=1/L=1 for examples 1 and 2. Further assume that variables v1 and v2 share the same x-axis.

1) 
```
yes? LET newv = v1[I=1:10] + v2[I=1:10]      !same dimension (10)
```

2) 
```
yes? LET newv = v1[I=1:10] + v2[I=5]         !newv has length of v1 (10)
```

3) We want to compare the salt values during the first half of the year with the values for the second half. Salt_diff will be placed on the time coordinates of the first variable—L=1:6. Ferret will issue a warning about ambiguous coordinates.

```
yes? LET salt_diff = salt[L=1:6] - salt[L=7:12]
```

4) In this example the variable zero will be promoted along each axis.

```
yes? LET zero = 0 * (i+j)
yes? LIST/I=1:5/J=1:5  zero          !5X5 matrix of 0's
```

5) Here we calculate density; salt and temp are on the same grid. This expression is an XYZ volume of points (100×100×10) of density at 10 depths based on temperature and salinity values at the top layer (K=1).

```
yes? SET REGION/I=1:100/J=1:100
yes? LET dens = rho_un (temp[K=1], salt[K=1], Z[G=temp,K=1:10]
```

## Ch3 Sec2.3. Functions

Functions are utilized with standard mathematical notation in Ferret. The arguments to functions are constants, constant arrays, pseudo-variables, and variables, possibly with associated qualifiers in square brackets, and expressions. Thus, all of these are valid function references:

- `EXP(-1)`
- `MAX(a,b)`
- `TAN(a/b)`
- `SIN(Y[g=my_sst])`
- `DAYS1900(1989,{3,6,9},1)`

A few functions also take strings as arguments. String arguments must be enclosed in double quotes. For example, a function to write variable "u" into a file named "my_output.v5d", formatted for the Vis5D program might be implemented as

- `LOAD WRITE_VIS5D("my_output.v5d", a)`

You can list function names and argument lists with:

```
yes? SHOW FUNCTIONS           ! List all functions
Yes? SHOW FUNCTIONS *TAN      ! List all functions containing string
```

Valid functions are:

## Ch3 Sec2.3.1. MAX

**MAX(VAR)** Compares two fields and selects the point by point maximum.
`MAX( temp[K=1], temp[K=2] )` returns the maximum temperature comparing the first 2 z-axis levels.

## Ch3 Sec2.3.2. MIN

**MIN(VAR)** Compares two fields and selects the point by point minimum.
`MIN( airt[L=10], airt[L=9] )` gives the minimum air temperature comparing two timesteps.

## Ch3 Sec2.3.3. INT

**INT (X)** Truncates values to integers.
`INT( salt )` returns the integer portion of variable "salt" for all values in the current region.

## Ch3 Sec2.3.4. ABS

**ABS(X)** absolute value.
`ABS( U )` takes the absolute value of U for all points within the current region

## Ch3 Sec2.3.5. EXP

**EXP(X)** exponential $e^x$; argument is real.
`EXP( X )` raises e to the power X for all points within the current region

## Ch3 Sec2.3.6. LN

**LN(X)** Natural logarithm $\log_e X$; argument is real.
`LN( X )` takes the natural logarithm of X for all points within the current region

## Ch3 Sec2.3.7. LOG

LOG(X) Common logarithm $\log_{10} X$; argument is real.
`LOG( X )` takes the common logarithm of X for all points within the current region

## Ch3 Sec2.3.8. SIN

**SIN(THETA)**Trigonometric sine; argument is in radians and is treated modulo 2*pi.
`SIN( X )` computes the sine of X for all points within the current region.

## Ch3 Sec2.3.9. COS

**COS(THETA )**Trigonometric cosine; argument is in radians and is treated modulo 2*pi.
`COS( Y )` computes the cosine of Y for all points within the current region

## Ch3 Sec2.3.10. TAN

TAN(THETA) Trigonometric tangent; argument is in radians and is treated modulo 2*pi.
`TAN( theta )` computes the tangent of theta for all points within the current region

## Ch3 Sec2.3.11. ASIN

**ASIN(X)** Trigonometric arcsine (-pi/2,pi/2) of X in radians.The result will be flagged as missing if the absolute value of the argument is greater than 1; result is in radians.
`ASIN( value )` computes the arcsine of "value" for all points within the current region

## Ch3 Sec2.3.12. ACOS

**COS(X)** Trigonometric arccosine (0,pi), in radians. The result will be flagged as missing of the absolute value of the argument greater than 1; result is in radians.
`ACOS ( value )` computes the arccosine of "value" for all points within the current region

## Ch3 Sec2.3.13. ATAN

**ATAN(X)** Trigonometric arctangent (-pi/2,pi/2); result is in radians.
`ATAN( value )` computes the arctangent of "value" for all points within the current region

## Ch3 Sec2.3.14. ATAN2

**ATAN2(X,Y)** 2-argument trigonometric arctangent of Y/X (-pi,pi); discontinuous at Y=0.
`ATAN2( X,Y )` computes the 2-argument arctangent of Y/X for all points within the current region

## Ch3 Sec2.3.15. MOD

**MOD(A,B)** Modulo operation ( arg1 – arg2*[arg1/arg2] ). Returns the remainder when the first argument is divided by the second.
`MOD( X,2 )` computes the remainder of X/2 for all points within the current region

## Ch3 Sec2.3.16. DAYS1900

**DAYS1900(year,month,day)** computes the number of days since 1 Jan 1900. This function is useful in converting dates to Julian days. If the year is prior to 1900 a negative number is returned. This means that it is possible to compute Julian days relative to, say, 1800 with the expression

```
LET jday1800 = DAYS1900 ( year, month, day) - DAYS1900( 1800,1,1)
```

## Ch3 Sec2.3.17. MISSING

**MISSING(A,B)** Replaces missing values in the first argument (multi-dimensional variable) with the second argument; the second argument may be any conformable variable.
`MISSING( temp, -999 )` replaces missing values in temp with –999
`MISSING( sst, temp[D=coads_climatology] )` replaces missing sst values with temperature from the COADS climatology

## Ch3 Sec2.3.18. IGNORE0

**IGNORE0(VAR)** Replaces zeros in a variable with the missing value flag for that variable.
`IGNORE0( salt )` replaces zeros in salt with the missing value flag

## Ch3 Sec2.3.19. RANDU

**RANDU(A)** Generates a grid of uniformly distributed [0,1] pseudo-random values. The first valid value in the field is used as the random number seed. Values that are flagged as bad remain flagged as bad in the random number field.
`RANDU( temp[I=105:135,K=1:5] )` generates a field of uniformly distributed random values of the same size and shape as the field "temp[I=105:135,K=1:5]" using temp[I=105,k=1] as the pseudo-random number seed.

## Ch3 Sec2.3.20. RANDN

**RANDN(A)** Generates a grid of normally distributed pseudo-random values. As above, but normally distributed rather than uniformly distributed.

## Ch3 Sec2.3.21. RHO_UN

**RHO_UN(SALT, TEMP, P)** Calculates rho (density kg/m^3) from salt (psu), temperature (deg C) and pressure (decibars) using the 1980 UNESCO International Equation of State (IES80). The routine uses the high pressure equation of state from Millero et al. (1980) and the oneatmosphere equation of state from Millero and Poisson (1981) as reported in Gill (1982). The notation follows Millero et al. (1980) and Millero and Poisson (1981).
`RHO_UN( salt, temp, Z )`

## Ch3 Sec2.3.22. THETA_FO

**THETA_FO(SALT, TEMP, Z, REF)** Calculates local potential temperature field at salt (psu), temperature (deg C), pressure (decibars) and specified reference pressure. This calculation uses Bryden (1973) polynomial for adiabatic lapse rate and Runge-Kutta 4th order integration algorithm. References: Bryden, H., 1973, Deep-Sea Res., 20, 401–408; Fofonoff, N.M, 1977, Deep-Sea Res., 24, 489–491.

```
THETA_FO( salt, temp, Z, Z_reference )
```

## Ch3 Sec2.3.23. RESHAPE

**REASHAPE(A, B)** The result of the RESHAPE function will be argument A "wrapped" on the grid of argument B. The limits given on argument 2 are used to specify subregions within the grid into which values should be reshaped.

```
RESHAPE(Tseries,MonthYear)
```

Two common uses of this function are to view multi-year time series data as a 2-dimensional field of 12-months vs. year and to map ABSTRACT axes onto real world coordinates. An example of the former is

```
DEFINE AXIS/t=15-JAN-1982:15-DEC-1985/NPOINTS=48/UNITS=DAYS tcal
LET my_time_series = SIN(T[gt=tcal]/100)
! reshape 48 months into a 12 months by 4 year matrix
DEFINE AXIS/t=1982:1986:1 tyear
DEFINE AXIS/Z=1:12:1 zmonth
LET out_grid = Z[GZ=zmonth]+T[GT=tyear]
LET my_reshaped = RESHAPE(my_time_series, out_grid)
SHOW GRID my_reshaped
   GRID (G001)
name       axis              # pts   start                end
normal     X
normal     Y
ZMONTH     Z                  12 r   1                    12
TYEAR      T                   5 r   1982                 1986
```

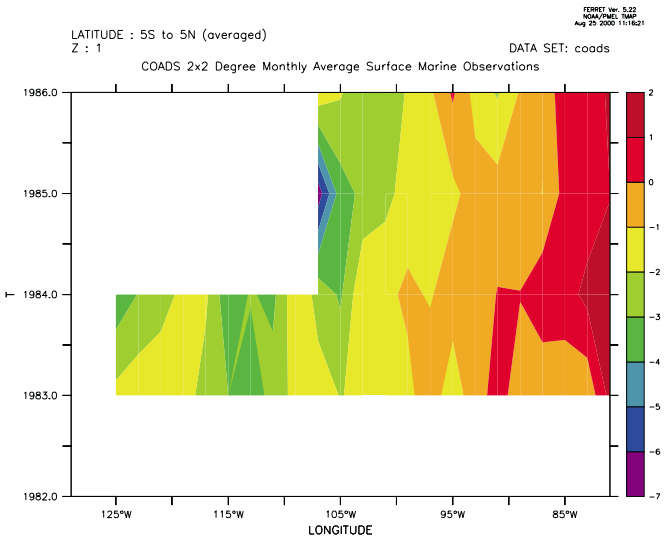
For any axis X,Y,Z, or T if the axis differs between the input output grids, then limits placed upon the region of the axis in argument two (the output grid) can be used to restrict the geometry into which the RESHAPE is performed. Continuing with the preceding example:

! Now restrict the output region to obtain a 6 month by 8 year matrix

```
LIST RESHAPE(my_time_series,out_grid[k=1:6])
         RESHAPE(MY_TIME_SERIES,OUT_GRID[K=1:6])
                1       2       3       4       5       6
                1       2       3       4       5       6
1982   / 1:  0.5144  0.7477  0.9123  0.9931  0.9827  0.8820
1983   / 2:  0.7003  0.4542  0.1665 -0.1366 -0.4271 -0.6783
1984   / 3: -0.8673 -0.9766 -0.9962 -0.9243 -0.7674 -0.5401
1985   / 4: -0.2632  0.0380  0.3356  0.6024  0.8138  0.9505
1986   / 5:  0.9999  0.9575  0.8270  0.6207  0.3573  0.0610
```

For any axis X,Y,Z, or T if the axis is the same in the input and output grids then the region from argument 1 will be preserved in the output. This implies that when the above technique is used on multi-dimensional input, only the axes which differ between the input and output grids are affected by the RESHAPE operation. The following filled contour plot of longitude by year number illustrates by expanding on the above example: (Figure 3_2)

```
!  The year-by-year progression January winds for a longitudinal patch
!  averaged from 5s to 5n across the eastern Pacific Ocean.  Note that
!  k=1 specifies January, since the Z axis is month
USE coads
LET out_grid = Z[GZ=zmonth]+T[GT=tyear]+X[GX=uwnd]+Y[GY=uwnd]
LET uwnd_mnth_ty = RESHAPE(uwnd, out_grid)
FILL uwnd_mnth_ty[X=130W:80W,Y=5S:5N@AVE,K=1]
```




RESHAPE(UWND, OUT_GRID)

Figure 3_2

In the second usage mentioned, to map ABSTRACT axes onto real world coordinates, suppose xpts and ypts contain time series of length Nt points representing longitude and latitude points along an oceanographic ship track and the variable global_sst contains global sea surface temperature data. Then the result of

LET sampled_sst = SAMPLEXY(global_sst, xpts, ypts)

will be a 1-dimensional grid: Nt points along the XABSTRACT axis. The RESHAPE function can be used to remap this data to the original time axis using

```
RESHAPE(sampled_sst, xpts)
LET sampled_sst = SAMPLEXY(global_sst,
Xpts[t=1-jan-1980:15-jan-1980],
ypts[t=1-jan-1980:15-jan-1980])
RESHAPE(sampled_sst, xpts[t=1-jan-1980:15-jan-1980])
```

When the input and output grids share any of the same axes, then the specified sub-region along those axes will be preserved in the RESHAPE operation. In the example "RESHAPE(myTseries,myMonthYearGrid)" this means that if myTseries and myMonthYearGrid were each multidimensional variables with the same latitude and longitude grids then

```
RESHAPE(myTseries[X=130E:80W,Y=5S:5N],myMonthYearGrid)
```

would map onto the X=130E:80W,Y=5S:5N sub-region of the grid of myMonthYearGrid. When the input and output axes differ the sub-region of the output that is utilized may be controlled by inserting explicit limit qualifiers on the second argument

## Ch3 Sec2.3.24. ZAXREPLACE

**ZAXREPLACE(V,ZVALS,ZAX)** Convert between alternative monotonic Zaxes, where the mapping between the source and destination Z axes is a function of X,Y, and or T. Typical applications in the field of oceanography include converting from a Z axis of layer number to a Z axis in units of depth (e.g., for sigma coordinate fields) and converting from a Z axes of depth to one of density (for a stably stratified fluid).

Argument 1, V, is the field of data values, say temperature on the "source" Z-axis, say, layer number. The second argument, ZVALS, contains values in units of the desired destination Z axis (ZAX) on the Z axis as V — for example, depth values associated with each vertical layer. The third argument, ZAX, is any variable defined on the destination Z axis, often "Z[gz=zaxis_name]" is used. For example:

Contour salt as a function of density:

```
yes? set dat ocean_atlas_annual
! Define density sigma, then density axis axden
yes? let sigma=rho_un(salt,temp,0)-1000
yes? define axis/z=21:28:.05 axden
! Regrid to density
yes? let saltonsigma= ZAXREPLACE( salt, sigma, z[gz=axdens])
! MakePacific plot
yes? fill/y=0/x=120e:75w/yli=28:21:-1 saltonsigma
```

## Ch3 Sec2.3.25. XSEQUENCE, YSEQUENCE, ZSEQUENCE, TSEQUENCE

**XSEQUENCE(A), YSEQUENCE(A), ZSEQUENCE(A), TSEQUENCE(A)** Unravels the data from the argument into a 1-dimensional line of data on an ABSTRACT axis.

## Ch3 Sec2.3.26. FFTA

**FFTA(A)** Computes fft amplitude spectra, normalized by 1/N

| | | |
|---|---|---|
| **Arguments:** | A | Variable with regular time axis. |
| **Result Axes:** | X | Inherited from A |
| | Y | Inherited from A |
| | Z | Inherited from A |
| | T | Generated by the function: frequency in cyc/(time units from A) |

See the demonstration script ef_fft_demo.jnl for an example using this function.

FFTA returns a(j) in

$$f(t) = \sum_{(j=1 \text{ to } N/2)} (j) \cos(j \quad t + \quad (j))]$$

where [ ] means "integer part of", =2 pi/T is the fundamental frequency, and T=N* t is the time span of the data input to FFTA. is the phase (returned by FFTP, see next section)

The units of the returned time axis are "cycles/ t" where t is the time increment.

Even and odd N's are allowed. N need not be a power of 2. FFTA and FFTP assume f(1)=f(N+1), and the user gives the routines the first N pts.

The code is based on the FFT routines in Swarztrauber's FFTPACK available at www.netlib.org.

## Ch3 Sec2.3.27. FFTP

FFTP(A) Computes fft phase

| | | |
|---|---|---|
| **Arguments:** | A | Variable with regular time axis. |
| **Result Axes:** | X | Inherited from A |
| | Y | Inherited from A |
| | Z | Inherited from A |
| | T | Generated by the function: frequency in cyc/(time units from A) |

See the demonstration script ef_fft_demo.jnl for an example using this function.

FFTP returns (j) in

f(t) = $_{(j=1 \text{ to } N/2)}$ (j) cos(j t + (j))]

where [ ] means "integer part of", =2 pi/T is the fundamental frequency, and T=N* t is the time span of the data input to FFTA.

The units of the returned time axis are "cycles/ t" where t is the time increment.

Even and odd N's are allowed. Power of 2 not required. FFTA and FFTP assume f(1)=f(N+1), and the user gives the routines the first N pts.

The code is based on the FFT routines in Swarztrauber's FFTPACK available at www.netlib.org.

## Ch3 Sec2.3.28. SAMPLEI

**SAMPLEI(TO_BE_SAMPLED,X_INDICES)** samples a field at a list of X indices, which are a subset of its X axis

| | | |
|---|---|---|
| **Arguments:** | TO_BE_SAMPLED | Data to sample |
| | X_INDICES | list of indices of the variable TO_BE_SAMPLED |
| **Result Axes:** | X | ABSTRACT; length same as X_INDICES |
| | Y | Inherited from TO_BE_SAMPLED |
| | Z | Inherited from TO_BE_SAMPLED |
| | T | Inherited from TO_BE_SAMPLED |

See the demonstration ef_sort_demo.jnl for a common useage of this function. As with other functions which change axes, specify any region information for the variable TO_BE_SAMPLED explicitly in the function call, e.g.

```
yes? LET sampled_data = samplei(airt[X=160E:180E], xindices)
```

## Ch3 Sec2.3.29. SAMPLEJ

**SAMPLEJ(TO_BE_SAMPLED,Y_INDICES)** samples a field at a list of Y indices, which are a subset of its Y axis

| | | |
|---|---|---|
| **Arguments:** | TO_BE_SAMPLED | Data to be sample |

| | | |
|---|---|---|
| | Y_INDICES | list of indices of the variable TO_BE_SAMPLED |
| **Result Axes:** | X | Inherited from TO_BE_SAMPLED |
| | Y | ABSTRACT; length same as Y_INDICES |
| | Z | Inherited from TO_BE_SAMPLED |
| | T | Inherited from TO_BE_SAMPLED |

See the demonstration ef_sort_demo.jnl for a common useage of this function. As with other functions which change axes, specify any region information for the variable TO_BE_SAMPLED explicitly in the function call.

## Ch3 Sec2.3.30. SAMPLEK

**SAMPLEK(TO_BE_SAMPLED, Z_INDICES)** samples a field at a list of Z indices, which are a subset of its Z axis

| | | |
|---|---|---|
| **Arguments:** | TO_BE_SAMPLED | Data to sample |
| | Z_INDICES | list of indices of the variable TO_BE_SAMPLED |
| **Result Axes:** | X | Inherited from TO_BE_SAMPLED |
| | Y | Inherited from TO_BE_SAMPLED |
| | Z | ABSTRACT; length same as Z_INDICES |
| | T | Inherited from TO_BE_SAMPLED |

See the demonstration ef_sort_demo.jnl for a common useage of this function. As with other functions which change axes, specify any region information for the variable TO_BE_SAMPLED explicitly in the function call.

## Ch3 Sec2.3.31. SAMPLEL

**SAMPLEL(TO_BE_SAMPLED, T_INDICES)** samples a field at a list of T indices, a subset of its T axis

| | | |
|---|---|---|
| **Arguments:** | TO_BE_SAMPLED | Data to sample |
| | T_INDICES | list of indices of the variable TO_BE_SAMPLED |
| **Result Axes:** | X | Inherited from TO_BE_SAMPLED |
| | Y | Inherited from TO_BE_SAMPLED |
| | Z | Inherited from TO_BE_SAMPLED |

| | | |
|---|---|---|
| | T | ABSTRACT; length same as X_INDICES |

See the demonstration ef_sort_demo.jnl for a common useage of this function. As with other functions which change axes, specify any region information for the variable TO_BE_SAMPLED explicitly in the function call.

## Ch3 Sec2.3.32. SAMPLEIJ

**SAMPLEIJ(DAT_TO_SAMPLE,XPTS,YPTS)** Returns data sampled at a subset of its grid points, defined by (XPTS, YPTS)

| | | |
|---|---|---|
| **Arguments:** | DAT_TO_SAMPLE | Data to sample, field of x, y, and perhaps z and t |
| | XPTS | X indices of grid points |
| | YPTS | Y indices of grid points |
| **Result Axes:** | X | ABSTRACT, length of list (xpts,ypts) |
| | Y | NORMAL (no axis) |
| | Z | Inherited from DAT_TO_SAMPLE |
| | T | Inherited from DAT_TO_SAMPLE |

## Ch3 Sec2.3.33. SAMPLET_DATE

**SAMPLET_DATE (DAT_TO_SAMPLE, YR, MO, DAY, HR, MIN, SEC)** Returns data sampled by interpolating to one or more times

| | | |
|---|---|---|
| **Arguments:** | DAT_TO_SAMPLE | Data to sample, field of x, y, z and t |
| | YR | Year(s), integer YYYY |
| | MO | Month(s), integer month number MM |
| | DAY | Day(s) of month, integer DD |
| | HR | Hour(s) integer HH |
| | MIN | Minute(s), integer MM |
| | SEC | Second(s), integer SS |
| **Result Axes:** | X | Inherited from DAT_TO_SAMPLE |
| | Y | Inherited from DAT_TO_SAMPLE |
| | Z | Inherited from DAT_TO_SAMPLE |
| | T | ABSTRACT; length is # times sampled |

**Example:**

List wind speed at a subset of points from the COADS data set

```
yes? USE coads
yes? SET REGION/X=131E:135E/Y=39N
yes? LET my_wspd =
samplet_date(wspd,{1986,1986},{5,8},{16,15},{12,12},{0,0},{0, 0})
yes? LIST my_wspd

SAMPLET_DATE(WSPD,{1986,1986},{5,8},{16,15},{12,12},{0,0},{0,0})
             LATITUDE: 39N
             DATA SET: /home/shiva/data/coads.cdf
           131E   133E   135E
            56     57     58
 1   / 1:  5.130  4.690  5.120
 2   / 2:  5.267  4.962  4.666
```

## Ch3 Sec2.3.34. SAMPLEXY

**SAMPLEXY(DAT_TO_SAMPLE,XPTS,YPTS)** Returns data sampled at a set of (X,Y) points, using linear interpolation

| | | |
|---|---|---|
| **Arguments:** | DAT_TO_SAMPLE | Data to sample |
| | XPTS | X values of sample points |
| | YPTS | Y values of sample points |
| **Result Axes:** | X | ABSTRACT; length same as XPTSand YPTS |
| | Y | NORMAL (no axis) |
| | Z | Inherited from DAT_TO_SAMPLE |
| | T | Inherited from DAT_TO_SAMPLE |

**Example:**

Plot a section of data taken along a slanted line in the Pacific (Figure3_3a). One could use a ship track, specifying its coordinates as xlon, ylat.

```
yes? USE ocean_atlas_annual
yes? LET xlon = 234.5 + I[I=1:50]       ! define the slant line
yes? LET dely = 24./49
yes? LET ylat = 24.5 - dely*i[i=1:50] + dely
yes? PLOT/VS/LINE/SYM=27 xlon,ylat      ! line off Central America
yes? GO land
```

Now sample the field "salt" along this track and make a filled contour plot. The horizontal axis is abstract; it is a count of the number of points along the track. (Figure3_3b)

```
yes? LET slantsalt = samplexy(salt,xlon,ylat)
yes? FILL/LEVELS=(33.2,35.2,0.1)/VLIMITS=0:4000 slantsalt
```

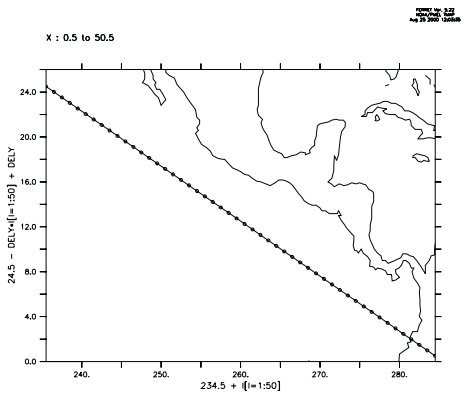



Figure3_3a

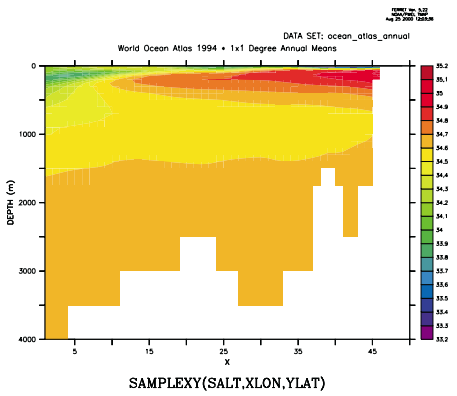



Figure3_3b

## Ch3 Sec2.3.35. SCAT2GRIDGAUSS_XY

**SCAT2GRIDGAUSS_XY(XPTS, YPTS, F, XAX, YAX, XSCALE, YSCALE, XCUTOFF, YCUTOFF)** Use Gaussian weighting to grid scattered data to an XY grid

| | | |
|---|---|---|
| **Arguments:** | XPTS | x-coordinates of scattered input triples; may be fcn of time |
| | YPTS | y-coordinates of scattered input triples; may be fcn of time |
| | F | F(X,Y) 3rd component of scattered input triples. May be fcn of time |
| | XAX | X-axis of output grid |
| | YAX | Y-axis of output grid |
| | XSCALE | Radius of influence in the X direction, in data units (e.g. lon or m) |
| | YSCALE | Radius of influence in the Y direction, in data units (e.g. lon or m) |
| | XCUTOFF | Cutoff for weight function in the X direction. Cutoff = 2 means the minimum weightweight is $e^{-4}$ |
| | YCUTOFF | Cutoff for weight function in the Y directionCutoff = 2 means the minimum weightweight is $e^{-4}$ |
| **Result Axes:** | X | Inherited from XAX |
| | Y | Inherited from YAX |
| | Z | NORMAL (no axis) |
| | T | Inherited from F |

The scat2gridgauss* functions use a Gaussian interpolation method map irregular locations ($x_n$, $y_n$) to a regular grid ($x_0$, $y_0$).

Parameters for a square grid and a fairly dense distribution of scattered points relative to the grid might be XSCALE=YSCALE = 0.5, and XCUTOFF=YCUTOFF = 2. To get better coverage, use a coarser grid or increase XSCALE, YSCALE and/or XCUTOFF, YCUTOFF.

The value of the gridded function F at each grid point ($x_0$, $y_0$) is computed by:

$$F(x_0,y_0) = \sum_{(n=1 \text{ to } Np)} F(x_n,y_n)W(x_n,y_n) \Big/ \sum_{(n=1 \text{ to } Np)} W(x_n,y_n)$$

Where Np is the total number of irregular points within the "influence region" of a particular grid point, (determined by the CUTOFF parameters, defined below). The Gaussian weight fucntion $W_n$ is given by

$$W_n(x_n,y_n) = \exp\{-[(x_n-x_0)^2/X^2 + (y_n-y_0)^2/Y^2]\}$$

X and Y in the denominators on the right hand side are the mapping scales, arguments 6 and 7.

The weight function has a nonzero value everywhere, so all of the scattered points in theory could be part of the sum for each grid point. To cut computation, the parameters XCUTOFF and YCUTOFFf are employed. If a cutoff of 2 is used, then scattered points with $W_n < e^{-4}$. This occurs where distances from the grid point are less than 2 times the mapping scales X or Y.

(Reference for this method: Kessler and McCreary, 1993: The Annual Wind-driven Rossby Wave in the Subthermocline Equatorial Pacivif, Journal of Physical Oceanography **23,** 1192 -1207)

## Ch3 Sec2.3.36. SCAT2GRIDGAUSS_XZ

**SCAT2GRIDGAUSS_XZ(XPTS, ZPTS, F, XAX, ZAX, XSCALE, ZSCALE, XCUTOFF, ZCUTOFF)** Use Gaussian weighting to grid scattered data to an XZ grid

| | | |
|---|---|---|
| **Arguments:** | XPTS | x-coordinates of scattered input triples; may be fcn of time |
| | ZPTS | z-coordinates of scattered input triples; may be fcn of time |
| | F | F(X,Z) 3rd component of scattered input triples. May be fcn of time |
| | XAX | X-axis of output grid |
| | ZAX | Z-axis of output grid |
| | XSCALE | Radius of influence in the X direction, in data units (e.g. lon or m) |
| | ZSCALE | Radius of influence in the Z direction, in data units (e.g. lon or m) |

| | | |
|---|---|---|
| | XCUTOFF | Cutoff for weight function in the X direction. Cutoff = 2 means the minimum weightweight is $e^{-4}$ |
| | ZCUTOFF | Cutoff for weight function in the Z direction. Cutoff = 2 means the minimum weightweight is $e^{-4}$ |
| **Result Axes:** | X | Inherited from XAX |
| | Y | NORMAL (no axis) |
| | Z | Inherited from ZAX |
| | T | Inherited from F |

See the discussion under SCAT2GRIDGAUSS_XY (p. 68)

## Ch3 Sec2.3.37. SCAT2GRIDGAUSS_XY

**SCAT2GRIDGAUSS_XY(YPTS, zPTS, F, YAX, ZAX, YSCALE, ZSCALE, YCUTOFF, ZCUTOFF)** Use Gaussian weighting to grid scattered data to a YZ grid

| | | |
|---|---|---|
| **Arguments:** | YPTS | y-coordinates of scattered input triples; may be fcn of time |
| | ZPTS | z-coordinates of scattered input triples; may be fcn of time |
| | F | F(Y,Z) 3rd component of scattered input triples. May be fcn of time |
| | YAX | Y-axis of output grid |
| | ZAX | Z-axis of output grid |
| | YSCALE | Radius of influence in the Y direction, in data units (e.g. lon or m) |
| | ZSCALE | Radius of influence in the Z direction, in data units (e.g. lon or m) |
| | YCUTOFF | Cutoff for weight function in the Y direction. Cutoff = 2 means the minimum weightweight is $e^{-4}$ |
| | ZCUTOFF | Cutoff for weight function in the Y directionCutoff = 2 means the minimum weightweight is $e^{-4}$ |
| **Result Axes:** | X | NORMAL (no axis) |
| | Y | Inherited from YAX |
| | Z | Inherited from ZAX |
| | T | Inherited from F |

See the discussion under SCAT2GRIDGAUSS_XY (p. 68)

## Ch3 Sec2.3.38. SCAT2GRIDLAPLACE_XY

**SCAT2GRIDLAPLACE_XY(XPTS, YPTS, F, XAX, YAX, CAY, NRNG)** Use Laplace/ Spline interpolation to grid scattered data to an XY grid.

| | | |
|---|---|---|
| **Arguments:** | XPTS | x-coordinates of scattered input triples. May be fcn of time |
| | YPTS | y-coordinates of scattered input triples. May be fcn of time |
| | F | F(X,Y) 3rd component of scattered input triples. May be fcn of time |
| | XAX | X-axis of output grid |
| | YAX | Y-axis of output grid |
| | CAY | Amount of spline eqation (between 0 and inf.) vs Laplace interpolation |
| | NRNG | Grid points more than NRNG grid spaces from the nearest data point are set to undefined. |
| **Result Axes:** | X | Inherited from XAX |
| | Y | Inherited from YAX |
| | Z | NORMAL (no axis) |
| | T | Inherited from F |

The SCAT2GRIDLAPLACE* functions employ the same interpolation method as is used by PPLUS, and appears elsewhere in Ferret, e.g. in contouring. The parameters are used as follows (quoted from the PPLUS Users Guide):

**CAY**
If CAY=0.0, Laplacian interpolation is used. The resulting surface tends to have rather sharp peaks and dips at the data points (like a tent with poles pushed up into it). There is no chance of spurious peaks appearing. As CAY is increased, Spline interpolation predominates over the Laplacian, and the surface passes through the data points more smoothly. The possibility of spurious peaks increases with CAY. CAY= infinity is pure Spline interpolation. An over relaxation process in used to perform the interpolation. A value of CAY=5 often gives a good surface.

**NRNG**
Any grid points farther than NRNG away from the nearest data point will be set to "undefined" The default used by PPLUS is NRNG = 5

## Ch3 Sec2.3.39. SCAT2GRIDLAPLACE_XZ

**SCAT2GRIDLAPLACE_XZ(XPTS, ZPTS, F, XAX, ZAX, CAY, NRNG)** Use Laplace/ Spline interpolation to grid scattered data to an XZ grid.

| | | |
|---|---|---|
| **Arguments:** | XPTS | x-coordinates of scattered input triples. May be fcn of time |
| | ZPTS | z-coordinates of scattered input triples. May be fcn of time |
| | F | F(X,Z) 3rd component of scattered input triples. May be fcn of time |
| | XAX | X-axis of output grid |
| | ZAX | Z-axis of output grid |
| | CAY | Amount of spline eqation (between 0 and inf.) vs Laplace interpolation |
| | NRNG | Grid points more than NRNG grid spaces from the nearest data point are set to undefined. |
| **Result Axes:** | X | Inherited from XAX |
| | Y | NORMAL (no axis) |
| | Z | Inherited from ZAX |
| | T | Inherited from F |

See the discussion under SCAT2GRIDLAPLACE_XY (p. 72)

## Ch3 Sec2.3.40. SCAT2GRIDLAPLACE_YZ

**SCAT2GRIDLAPLACE_YZ(YPTS, ZPTS, F, YAX, ZAX, CAY, NRNG)** Use Laplace/ Spline interpolation to grid scattered data to an YZ grid.

| | | |
|---|---|---|
| **Arguments:** | YPTS | y-coordinates of scattered input triples. May be fcn of time |
| | ZPTS | z-coordinates of scattered input triples. May be fcn of time |
| | F | F(Y,Z) 3rd component of scattered input triples. May be fcn of time |
| | YAX | Y-axis of output grid |
| | ZAX | Z-axis of output grid |
| | CAY | Amount of spline eqation (between 0 and inf.) vs Laplace interpolation |
| | NRNG | Grid points more than NRNG grid spaces from the nearest data point are set to undefined. |

| | | |
|---|---|---|
| **Result Axes:** | X | NORMAL (no axis) |
| | Y | Inherited from YAX |
| | Z | Inherited from ZAX |
| | T | Inherited from F |

See the discussion under SCAT2GRIDLAPLACE_XY (p. 73)

## Ch3 Sec2.3.41. SORTI

**SORTI(DAT):** Returns indices of data, sorted on the I axis in increasing order

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to sort |
| **Result Axes:** | X | ABSTRACT, same length as DAT x-axis |
| | Y | Inherited from DAT |
| | Z | Inherited from DAT |
| | T | Inherited from DAT |

SORTI, SORTJ, SORTK, and SORTL return the indices of the data after it has been sorted. These functions are used in conjunction with functions such as the SAMPLE functions to do sorting and sampling. See the demonstration ef_sort_demo.jnl for common useage of these functions.

## Ch3 Sec2.3.42. SORTJ

**SORTJ(DAT)** Returns indices of data, sorted on the I axis in increasing order

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to sort |
| **Result Axes:** | X | Inherited from DAT |
| | Y | ABSTRACT, same length as DAT y-axisInherited from DAT |
| | Z | Inherited from DAT |
| | T | Inherited from DAT |

SORTI, SORTJ, SORTK, and SORTL return the indices of the data after it has been sorted. These functions are used in conjunction with functions such as the SAMPLE functions to do sorting and sampling. See the demonstration ef_sort_demo.jnl for common useage of these functions.

## Ch3 Sec2.3.43. SORTK

**SORTK(DAT)** Returns indices of data, sorted on the I axis in increasing order

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to sort |
| **Result Axes:** | X | Inherited from DAT |
| | Y | Inherited from DAT |
| | Z | ABSTRACT, same length as DAT x-axis |
| | T | Inherited from DAT |

SORTI, SORTJ, SORTK, and SORTL return the indices of the data after it has been sorted. These functions are used in conjunction with functions such as the SAMPLE functions to do sorting and sampling. See the demonstration ef_sort_demo.jnl for common useage of these functions.

## Ch3 Sec2.3.44. SORTL

**SORTL(DAT)** Returns indices of data, sorted on the L axis in increasing order

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to sort |
| **Result Axes:** | X | Inherited from DAT |
| | Y | Inherited from DAT |
| | Z | Inherited from DAT |
| | T | ABSTRACT, same length as DAT x-axis |

SORTI, SORTJ, SORTK, and SORTL return the indices of the data after it has been sorted. These functions are used in conjunction with functions such as the SAMPLE functions to do sorting and sampling. See the demonstration ef_sort_demo.jnl for common useage of these functions.

## Ch3 Sec2.3.45. TAUTO_COR

**TAUTO_COR:** (beta) Compute autocorrelation function (ACF) of time series, lags of 0,...,N-1

| | | |
|---|---|---|
| **Arguments:** | A | A function of time |
| **Result Axes:** | X | Inherited from DAT |
| | Y | Inherited from DAT |
| | Z | Inherited from DAT |

| | |
|---|---|
| T | ABSTRACT, same length as A tim axis (lags) |

## Ch3 Sec2.4. Transformations

Transformations (e.g., averaging, integrating, etc.) may be specified along the axes of a variable. Some transformations (e.g., averaging) reduce a range of data to a point; others (e.g., differentiating) retain the range.

When transformations are specified along more than one axis of a single variable the order of execution is X axis first, then Y then Z then T.

The regridding transformations are described in the chapter “Grids and Regions” (p. 101).

Example syntax: `TEMP[Z=0:100@LOC:20]` (depth at which temp has value 20)

Valid transformations are

| Transform | Default Argument | Description |
|---|---|---|
| @DIN | | definite integral (weighted sum) |
| @IIN | | indefinite integral (weighted running sum) |
| @AVE | | average |
| @VAR | | unweighted variance |
| @MIN | | minimum |
| @MAX | | maximum |
| @SHF | 1 pt | shift |
| @SBX | 3 pt | boxcar smoothed |
| @SBN | 3 pt | binomial smoothed |
| @SHN | 3 pt | Hanning smoothed |
| @SPZ | 3 pt | Parzen smoothed |
| @SWL | 3 pt | Welch smoothed |
| @DDC | | centered derivative |
| @DDF | | forward derivative |
| @DDB | | backward derivative |
| @NGD | | number of valid points |
| @NBD | | number of bad (invalid) points flagged |
| @SUM | | unweighted sum |
| @RSUM | | running unweighted sum |
| @FAV | 3 pt | fill missing values with average |
| @FLN:n | 1 pt | fill missing values by linear interpolation |
| @FNR:n | 1 pt | fill missing values with nearest point |

| Transform | Default Argument | Description |
|---|---|---|
| @LOC | 0 | coordinate of ... (e.g., depth of 20 degrees) |
| @WEQ | | "weighted equal" (integrating kernel) |
| @CDA | | closest distance above |
| @CDB | | closest distance below |
| @CIA | | closest index above |
| @CIB | | closest index below |

The command SHOW TRANSFORM will produce a list of currently available transformations.

**Examples: Transformations**

```
U[Z=0:100@AVE]          – average of u between 0 and 100 in Z
sst[T=@SBX:10]          – box-car smooths sst with a 10 time point filter
tau[L=1:25@DDC]         – centered time derivative of tau
v[L=@IIN]               – indefinite (accumulated) integral of v
qflux[X=@AVE,Y=@AVE]    – XY area-averaged qflux
```

## Ch3 Sec2.4.1. General information about transformations

Transformations are normally computed axis by axis; if multiple axes have transformations specified simultaneously (e.g., `U[Z=@AVE,L=@SBX:10]`) the transformations will be applied sequentially in the order X then Y then Z then T. There are two exceptions to this: if @DIN is applied simultaneously to both the X and Y axes (in units of degrees of longitude and latitude, respectively) the calculation will be carried out on a per-unit-area basis (as a true double integral) instead of a per-unit-length basis, sequentially. This ensures that the COSINE(latitude) factors will be applied correctly. The same applies to @AVE simultaneously on X and Y.

Data that are flagged as invalid are excluded from calculations.

When calculating integrals and derivatives (@IIN, @DIN, @DDC, @DDF, and @DDB) Ferret attempts to use standardized units for the grid coordinates. If the underlying axis is in a known unit of length Ferret converts grid box lengths to meters. If the underlying axis is in a known unit of time Ferret converts grid box lengths to seconds. If the underlying axis is degrees of longitude a factor of COSINE (latitude) is applied to the grid box lengths in meters.

If the underlying axis units are unknown Ferret uses those unknown units for the grid box lengths. (If Ferret does not recognize the units of an axis it displays a message to that effect when the DEFINE AXIS or SET DATA command defines the axis.) See command DEFINE AXIS/UNITS (p. 255) in the Commands Reference in this manual for a list of recognized units.

All integrations and averaging are accomplished by multiplying the width of each grid box by the value of the variable in that grid box—then summing and dividing as appropriate for the particular transformation.

If integration or averaging limits are given as world coordinates, the grid boxes at the edges of the region specified are weighted according to the fraction of grid box that actually lies within the specified region. If the transformation limits are given as subscripts, the full box size of each grid point along the axis is used—including the first and last subscript given. The region information that is listed with the output reflects this.

Some transformations (derivatives, shifts, smoothers) require data points from beyond the edges of the indicated region in order to perform the calculation. Ferret automatically accesses this data as needed. It flags edge points as missing values if the required beyond-edge points are unavailable (e.g., @DDC applied on the X axis at I=1).

## Ch3 Sec2.4.2. Transformations applied to irregular regions

Since transformations are applied along the orthogonal axes of a grid they lend themselves naturally to application over "rectangular" regions (possibly in 3 or 4 dimensions). Ferret has sufficient flexibility, however, to perform transformations over irregular regions.

Suppose, for example, that we wish to determine the average wind speed within an irregularly shaped region of the globe defined by a threshold sea surface temperature value. We can do this through the creation of a mask, as in this example:

```
yes? SET DATA coads_climatology
yes? SET REGION/l=1/@t        ! January in the Tropical Pacific
yes? LET sst28_mask = IF sst GT 28 THEN 1
yes? LET masked_wind_speed = wspd * sst28_mask
yes? LIST masked_wind_speed[X=@AVE,Y=@AVE]
```

The variable sst28_mask is a collection of 1's and missing values. Using it as a multiplier on the wind speed field produces a new result that is undefined except in the domain of interest.

When using masking be aware of these considerations:

- Use undefined values rather than zeros to avoid contaminating the calculation with zero values.

- The masked region is composed of rectangles at the level of resolution of the gridded variables; the mask does NOT follow smooth contour lines. To obtain a smoother mask it may be desirable to regrid the calculation to a finer grid.

- Variables from different data sets can be used to mask one another. For example, the ETOPO60 bathymetry data set can be used to mask regions of land and sea.

## Ch3 Sec2.4.3. General information about smoothing transformations

Ferret provides several transformations for smoothing variables (removing high frequency variability). These transformations replace each value on the grid to which they are applied with a weighted average of the surrounding data values along the axis specified. For example, the expression u[T=@SPZ:3] replaces the value at each (I,J,K,L) grid point of the variable "u" with the weighted average

```
u at t = 0.25*(u at t-1) + 0.5*(u at t) + 0.25*(u at t+1)
```

The various choices of smoothing transformations (@SBX, @SBN, @SPZ, @SHN, @SWL) represent different shapes of weighting functions or "windows" with which the original variable is convolved. New window functions can be obtained by nesting the simple ones provided. For example, using the definitions

```
yes? LET ubox = u[L=@SBX:15]
yes? LET utaper = ubox[L=@SHN:7]
```

produces a 21-point window whose shape is a boxcar (constant weight) with COSINE (Hanning) tapers at each end.

Ferret may be used to directly examine the shape of any smoothing window: Mathematically, the shape of the smoothing window can be recovered as a variable by convolving it with a delta function. In the example below we examine (PLOT) the shape of a 15-point Welch window (Figure 3_4).

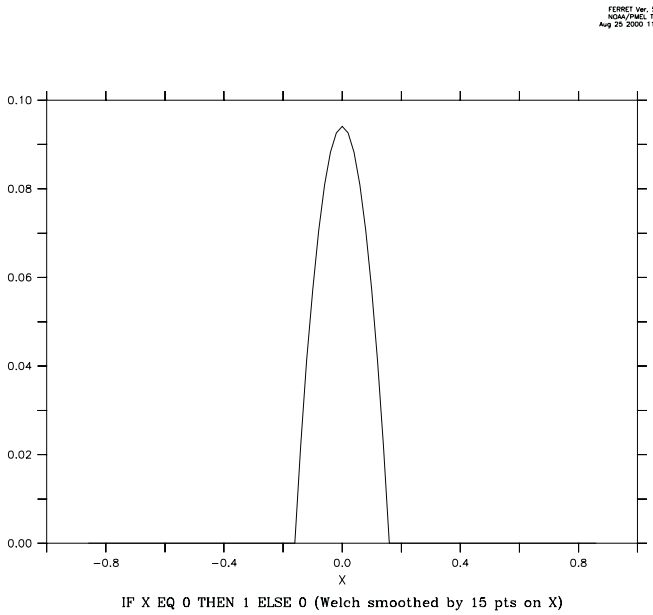



Figure 3_4

```
! define X axis as [-1,1] by 0.2
yes? GO unit_square
yes? SET REGION/X=-1:1
yes? LET delta =
```

```
      IF X EQ 0 THEN 1 ELSE 0
! convolve delta with Welch window
yes? PLOT delta[I=@SWL:15]
```

## Ch3 Sec2.4.4. @DIN—definite integral

The transformation @DIN computes the definite integral—a single value that is the integral between two points along an axis (compare with @IIN). It is obtained as the sum of the grid_box*variable product at each grid point. Grid points at the ends of the indicated range are weighted by the fraction of the grid box that falls within the integration interval.

If @DIN is specified simultaneously on multiple axes the calculation will be performed as a multiple integration rather than as sequential single integrations. The output will document this fact by indicating a transformation of "@IN4" or "XY integ." See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? CONTOUR/X=160E:160W/Y=5S:5N u[Z=0:50@DIN]
```

In a latitude/longitude coordinate system X=@DIN is sensitive to the COS(latitude) correction.

Integration over complex regions in space may be achieved by masking the multi-dimensional variable in question and using the multi-dimensional form of @DIN. For example

```
yes? LET salinity_where_temp_gt_15 = IF temp GT 15 THEN salt
yes?  LIST salinity_where_temp_gt_15[X=@DIN,Y=@DIN,Z=@DIN]
```

## Ch3 Sec2.4.5. @IIN—indefinite integral

The transformation @IIN computes the indefinite integral—at each subscript of the result it is the value of the integral from the start value to the upper edge of that grid box. It is obtained as a running sum of the grid_box*variable product at each grid point. Grid points at the ends of the indicated range are weighted by the fraction of the grid box that falls within the integration interval. See General Information (p 76) for important details about this transformation.

**Example**

```
yes? CONTOUR/X=160E:160W/Z=0 u[Y=5S:5N@IIN]
```

**Note 1:** The indefinite integral is always computed in the increasing coordinate direction. To compute the indefinite integral in the reverse direction use

```
LET reverse_integral = my_var[Y=lo:hi@DIN] - my_var[X=lo:hi@IIN]
```

**Note 2:** In a latitude/longitude coordinate system X=@IIN is sensitive to the COS(latitude) correction.

**Note 3:** The result of the indefinite integral is shifted by 1/2 of a grid cell from its "proper" location. This is because the result at each grid cell includes the integral computed to the upper end of that cell. (This was necessary in order that var[I=lo:hi@DIN] and var[I=lo:hi@IIN] produce consistent results.)

To illustrate, consider these commands

```
yes? LET one = x-x+1
yes? LIST/I=1:3 one[I=@din]
             X-X+1
             X: 0.5 to 3.5 (integrated)
          3.000
yes? LIST/I=1:3 one[I=@iin]
             X-X+1
             indef. integ. on X
 1   / 1:  1.000
 2   / 2:  2.000
 3   / 3:  3.000
```

The grid cell at I=1 extends from 0.5 to 1.5. The value of the integral at 1.5 is 1.000 as reported but the coordinate listed for this value is 1 rather than 1.5. Two methods are available to correct for this 1/2 grid cell shift.

**Method 1:**correct the result by subtracting the 1/2 grid cell error

```
yes? LIST/I=1:3 one[I=@iin] - one/2
             ONE[I=@IIN] - ONE/2
 1   / 1:  0.500
 2   / 2:  1.500
 3   / 3:  2.500
```

**Method 2:**correct the coordinates by shifting the axis 1/2 of a grid cell

```
yes? DEFINE AXIS/X=1.5:3.5:1 xshift
yes? LET SHIFTED_INTEGRAL =  one[I=@IIN]
yes? LET corrected_integral = shifted_integral[GX=xshift@ASN]
yes? LIST/I=1:3 corrected_integral
             SHIFTED_INTEGRAL[GX=XSHIFT@ASN]
 1.5 / 1:  1.000
 2.5 / 2:  2.000
 3.5 / 3:  3.000
```

## Ch3 Sec2.4.6. @AVE—average

The transformation @AVE computes the average weighted by grid box size—a single number representing the average of the variable between two endpoints.

If @AVE is specified simultaneously on multiple axes the calculation will be performed as a multiple integration rather than as sequential single integrations. The output will document this fact by showing @AV4 or "XY ave" as the transformation. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? CONTOUR/X=160E:160W/Y=5S:5N u[Z=0:50@AVE]
```

Note that the unweighted mean can be calculated using the @SUM and @NGD transformations.

Averaging over complex regions in space may be achieved by masking the multi-dimensional variable in question and using the multi-dimensional form of @AVE. For example

```
yes? LET salinity_where_temp_gt_15 = IF temp GT 15 THEN salt
yes? LIST salinity_where_temp_gt_15[X=@AVE,Y=@AVE,Z=@AVE]
```

## Ch3 Sec2.4.7. VAR—weighted variance

The transformation @VAR computes the weighted variance of the variable with respect to the indicated region (ref. Numerical Recipes, The Art of Scientific Computing, by William H. Press et al., 1986).

As with @AVE, if @VAR is applied simultaneously to multiple axes the calculation is performed as the variance of a block of data rather than as nested 1-dimensional variances. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.8. MIN—minimum

The transformation @MIN finds the minimum value of the variable within the specified axis range. See General Information (p 76) for important details about this transformation.

**Example**

For fixed Z and Y

```
yes? PLOT/T="1-JAN-1982":"1-JAN-1983"    temp[X=160E:160W@MIN]
```

plots a time series of the minimum temperature found between longitudes 160 east and 160 west.

## Ch3 Sec2.4.9. @MAX—maximum

The transformation @MAX finds the maximum value of the variable within the specified axis range. See also @MIN. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.10. @SHF:n—shift

The transformation @SHF shifts the data up or down in subscript by the number of points given as the argument. See General Information (p 76) for important details about this transformation.

**Examples:**

```
U[L=@SHF:2]
```

associates the value of U[L=3] with the subscript L=1.

```
U[L=@SHF:1]-U
```

gives the forward difference of the variable U along the L axis.

## Ch3 Sec2.4.11. @SBX:n—boxcar smoother

The transformation @SBX applies a boxcar window (running mean) to smooth the variable along the indicated axis. The width of the boxcar is the number of points given as an argument to the transformation. All points are weighted equally, regardless of the sizes of the grid boxes, making this transformation best suited to axes with equally spaced points. If the number of points specified is even, however, @SBX weights the end points of the boxcar smoother as ½.. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? PLOT/X=160W/Y=0 u[L=1:120@SBX:5]
```

The transformation @SBX does not reduce the number of points along the axis; it replaces each of the original values with the average of its surrounding points. Regridding can be used to reduce the number of points.

## Ch3 Sec2.4.12. @SBN:n—binomial smoother

The transformation @SBN applies a binomial window to smooth the variable along the indicated axis. The width of the smoother is the number of points given as an argument to the transformation. The weights are applied without regard to the widths of the grid boxes, making this

transformation best suited to axes with equally spaced points. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? PLOT/X=160W/Y=0/Z=0 u[L=1:120@SBN:15]
```

The transformation @SBN does not reduce the number of points along the axis; it replaces each of the original values with a weighted sum of its surrounding points. Regridding can be used to reduce the number of points. The argument specified with @SBN, the number of points in the smoothing window, must be an odd value; an even value would result in an effective shift of the data along its axis.

## Ch3 Sec2.4.13. @SHN:n — Hanning smoother

Transformation @SHN applies a Hanning window to smooth the variable along the indicated axis (ref. Numerical Recipes, The Art of Scientific Computing, by William H. Press et al., 1986). In other respects it is identical in function to the @SBN transformation. Note that the Hanning window used by Ferret contains only non-zero weight values with the window width. Some interpretations of this window function include zero weights at the end points. Use an argument of N-2 to achieve this effect (e.g., @SBX:5 is equivalent to a 7-point Hanning window which has zeros as its first and last weights). See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.14. @SPZ:n — Parzen smoother

Transformation @SPZ applies a Parzen window to smooth the variable along the indicated axis (ref. Numerical Recipes, The Art of Scientific Computing, by William H. Press et al., 1986). In other respects it is identical in function to the @SBN transformation. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.15. @SWL:n — Welch smoother

Transformation @SWL applies a Welch window to smooth the variable along the indicated axis (ref. Numerical Recipes, The Art of Scientific Computing, by William H. Press et al., 1986). In other respects it is identical in function to the @SBN transformation. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.16. @DDC—centered derivative

The transformation @DDC computes the derivative with respect to the indicated axis using a centered differencing scheme. The units of the underlying axis are treated as they are with integrations. If the points of the axis are unequally spaced, note that the calculation used is still $(F_{i+1} - F_{i-1}) / (X_{i+1} - X_{i-1})$ . See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? PLOT/X=160W/Y=0/Z=0 u[L=1:120@DDC]
```

## Ch3 Sec2.4.17. @DDF—forward derivative

The transformation @DDF computes the derivative with respect to the indicated axis. A forward differencing scheme is used. The units of the underlying axis are treated as they are with integrations. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? PLOT/X=160W/Y=0/Z=0 u[L=1:120@DDF]
```

## Ch3 Sec2.4.18. @DDB—backward derivative

The transformation @DDF computes the derivative with respect to the indicated axis. A backward differencing scheme is used. The units of the underlying axis are treated as they are with integrations. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? PLOT/X=160W/Y=0/Z=0 u[L=1:120@DDB]
```

## Ch3 Sec2.4.19. @NGD—number of good points

The transformation @NGD computes the number of good (valid) points of the variable with respect to the indicated axis. Use @NGD in combination with @SUM to determine the number of good points in a multi-dimensional region.

Note that, as with @VAR, when @NGD is applied simultaneously to multiple axes the calculation is applied to the entire block of values rather than to the individual axes. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.20. @NBD—number of bad points

The transformation @NBD computes the number of bad (invalid) points of the variable with respect to the indicated axis. Use @NBD in combination with @SUM to determine the number of bad points in a multi-dimensional region.

Note that, as with @VAR, when @NBD is applied simultaneously to multiple axes the calculation is applied to the entire block of values rather than to the individual axes. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.21. @SUM—unweighted sum

The transformation @SUM computes the unweighted sum (arithmetic sum) of the variable with respect to the indicated axis. This transformation is most appropriate for regions specified by subscript. If the region is specified in world coordinates, the edge points are not weighted—they are wholly included in or excluded from the calculation, depending on the location of the grid points with respect to the specified limits. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.22. @RSUM—running unweighted sum

The transformation @RSUM computes the running unweighted sum of the variable with respect to the indicated axis. @RSUM is to @IIN as @SUM is to @DIN. The treatment of edge points is identical to @SUM. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.23. @FAV:n—averaging filler

The transformation @FAV fills holes (values flagged as invalid) in variables with the average value of the surrounding grid points along the indicated axis. The width of the averaging window is the number of points given as an argument to the transformation. All of the surrounding points are weighted equally, regardless of the sizes of the grid boxes, making this transformation best suited to axes with equally spaced points. If the number of points specified is even, however, @FAV weights the end points of the filling region by 1/2. If any of the surrounding points are invalid they are omitted from the calculation. If all of the surrounding points are invalid the hole is not filled. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? CONTOUR/X=160W:160E/Y=5S:0 u[X=@FAV:5]
```

## Ch3 Sec2.4.24. @FLN:n — linear interpolation filler

The transformation @FLN:n fills holes in variables with a linear interpolation from the nearest non-missing surrounding point. n specifies the number of points beyond the edge of the indicated axis limits to include in the search for interpolants (default n = 1). Unlike @FAV, @FLN is sensitive to unevenly spaced points and computes its linear interpolation based on the world coordinate locations of grid points.

Any gap of missing values that has a valid data point on each end will be filled, regardless of the length of the gap. However, when a sub-region from the full span of the data is requested sometimes a fillable gap crosses the border of the requested region. In this case the valid data point from which interpolation should be computed is not available. The parameter n tells Ferret how far beyond the border of the requested region to look for a valid data point. See General Information (p 76) for important details about this transformation.

Example: To allow data to be filled only when gaps in i are less than 15 points, use the @CIA and @CIB transformations which return the distance from the nearest valid point.

```
yes? USE my_data
yes? LET allowed_gap = 15
yes? LET gap_size = my_var[i=@cia] + my_var[i=@cib]
yes? LET gap_mask = IF gap_size LE gap_allowed THEN 1
yes? LET my_answer = my_var[i=@fln) * gap_mask
```

## Ch3 Sec2.4.25. @FNR:n — nearest neighbor filler

The transformation @FNR:n is similar to @FLN:n, except that it replicates the nearest point to the missing value. In the case of points being equally spaced around the missing point, the mean value is used. See General Information (p 76) for important details about this transformation.

## Ch3 Sec2.4.26. @LOC — location of

The transformation @LOC accepts an argument value—the default value is zero if no argument is specified. The transformation @LOC finds the single location at which the variable first assumes the value of the argument. The result is in units of the underlying axis. Linear interpolation is used to compute locations between grid points. If the variable does not assume the value of the argument within the specified region the @LOC transformation returns an invalid data flag.

For example, temp[Z=0:200@LOC:18] finds the location along the Z axis (often depth in meters) at which the variable "temp" (often temperature) first assumes the value 18, starting at Z=0 and searching to Z=200. See General Information (p 76) for important details about this transformation.

```
yes? CONTOUR/X=160E:160W/Y=10S:10N    temp[Z=0:200@LOC:18]
```

produces a map of the depth of the 18-degree isotherm. See also the General Information about transformations section in this chapter (p. 76).

Note that the transformation @LOC can be used to locate non-constant values, too, as the following example illustrates:

**Example: locating non-constant values**

Determine the depth of maximum salinity.

```
yes? LET max_salt = salt[Z=@MAX]
yes? LET zero_at_max = salt - max_salt
yes? LET depth_of_max = zero_at_max[Z=@LOC:0]
```

## Ch3 Sec2.4.27. @WEQ—weighted equal; integration kernel

The @WEQ ("weighted equal") transformation is the subtlest and arguably the most powerful transformation within Ferret. It is a generalized version of @LOC; @LOC always determines the value of the axis coordinate (the variable X, Y, Z, or T) at the points where the gridded field has a particular value. More generally, @WEQ can be used to determine the value of any variable at those points. See General Information (p 76) for important details about this transformation.

Like @LOC, the transformation @WEQ finds the location along a given axis at which the variable is equal to the given (or default) argument. For example, V1[Z=@WEQ:5] finds the Z locations at which V1 equals "5". But whereas @LOC returns a single value (the linearly interpolated axis coordinate values at the locations of equality) @WEQ returns instead a field of the same size as the original variable. For those two grid points that immediately bracket the location of the argument, @WEQ returns interpolation coefficients. For all other points it returns missing value flags. If the value is found to lie identically on top of a grid point an interpolation coefficient of 1 is returned for that point alone. The default argument value is 0.0 if no argument is specified.

**Example 1**

```
yes? LET v1 = X/4
yes? LIST/X=1:6 v1, v1[X=@WEQ:1], v1[X=@WEQ:1.2]

 X      v1     @WEQ:1   @WEQ:1.2
___   _____   _______  _________

 1:  0.250     ....       ....
 2:  0.500     ....       ....
 3:  0.750     ....       ....
 4:  1.000    1.000     0.2000
 5:  1.250     ....     0.8000
 6:  1.500     ....       ....
```

The resulting field can be used as an "integrating kernel," a weighting function that when multiplied by another field and integrated will give the value of that new field at the desired location.

**Example 2**

Using variable v1 from the previous example, suppose we wish to know the value of the function X^2 (X squared) at the location where variable v1 has the value 1.2. We can determine it as follows:

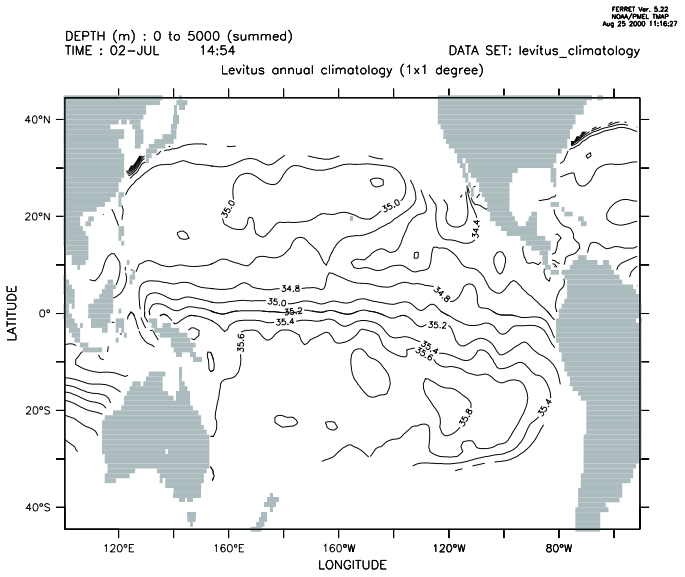
```
yes? LET x_squared = X^2
yes? LET integrand = x_squared * v1[X=@WEQ:1.2]
yes? LIST/X=1:6 integrand[X=@SUM]   !Ferret output below
           X_SQUARED * V1[X=@WEQ:1.2]
           X: 1 to 6 (summed)
         23.20
```

Notice that 23.20 = 0.8 * (5^2) + 0.2 * (4^2)

Below are two "real world" examples that produce fully labeled plots.

**Example 3: salinity on an isotherm**

Use the Levitus climatology to contour the salinity of the Pacific Ocean along the 20-degree isotherm (Figure 3_5).




Salinity on the 20 degree isotherm

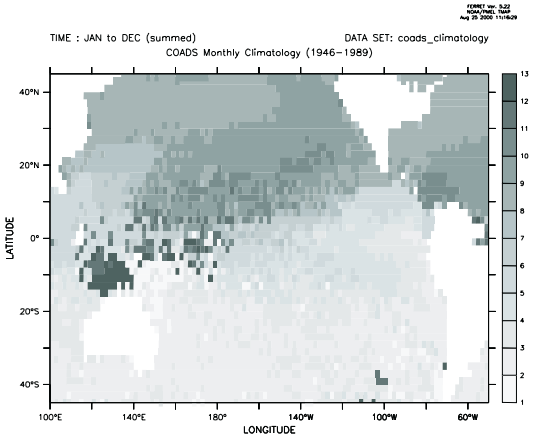
Figure 3_5

```
yes? SET DATA levitus_climatology       ! annual sub-surface
climatology
yes? SET REGION/X=100E:50W/Y=45S:45N    ! Pacific Ocean
yes? LET isotherm_20 = temp[Z=@WEQ:20]  ! depth kernel for 20 degrees
yes? LET integrand_20 = salt * isotherm_20
```

```
yes? SET VARIABLE/TITLE="Salinity on the 20 degree isotherm"
integrand_20
yes? PPL CONSET .12                          !contour label size (def. .08)
yes? CONTOUR/LEV=(33,37,.2) integrand_20[Z=@SUM]
yes? GO fland                                !continental fill
```

**Example 4: month with warmest sea surface temperatures**

Use the COADS data set to determine the month in which the SST is warmest across the Pacific Ocean. In this example we use the same principles as above to create an integrating kernel on the time axis. Using this kernel we determine the value of the time step index (which is also the month number, 1–12) at the time of maximum SST (Figure 3_6).




Month of warmest SST

Figure 3_6

```
yes? SET DATA coads_climatology        ! monthly surface climatology
yes? SET REGION/X=100E:50W/Y=45S:45N   ! Pacific Ocean
yes? SET MODE CAL:MONTH
yes? LET zero_at_warmest = sst - sst[l=@max]
yes? LET integrand = L[G=sst] * zero_at_warmest[L=@WEQ]  ! "L" is 1 to
12
yes? SET VARIABLE/TITLE="Month of warmest SST" integrand
yes? SHADE/L=1:12/PAL=inverse_grayscale integrand[L=@SUM]
```

## Ch3 Sec2.4.28. @ITP—interpolate

The @ITP transformation provides the same linear interpolation calculation that is turned on modally with SET MODE INTERPOLATE but with a higher level of control, as @ITP can be applied selectively to each axis. @ITP may be applied only to point locations along an axis. The result is the linear interpolation based on the adjoining values. See General Information (p 76) for important details about this transformation.

For example, for a Z axis with points at Z=0, 10, 20, ...

`V[Z=4@ITP]` will compute `0.6 * V[Z=0] + 0.4 * V[Z=10]`

## Ch3 Sec2.4.29. @CDA—closest distance above

The transformation @CDA will compute at each grid point how far it is to the closest valid point above this coordinate position on the indicated axis. The distance will be reported in the units of the axis. If a given grid point is valid (not missing) then the result of @CDA for that point will be 0.0. See the example for @CDB below. The result's units are now axis units, e.g., degrees of longitude to the next valid point above. See General Information (p 76) for important details about this transformation, and see the example under @CDB below (p 90).

## Ch3 Sec2.4.30. @CDB—closest distance below

The transformation @CDB will compute at each grid point how far it is to the closest valid point below this coordinate position on the indicated axis. The distance will be reported in the units of the axis. If a given grid point is valid (not missing) then the result of @CDB for that point will be 0.0. The result's units are now axis units, e.g., degrees of longitude to the next valid point below. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? USE coads_climatology
yes? SET REGION/x=125w:109w/y=55s/l=1
yes? LIST sst, sst[x=@cda], sst[x=@cdb]   ! results below

  Column 1: SST is SEA SURFACE TEMPERATURE (Deg C)
  Column 2: SST[X=@CDA:1] is SEA SURFACE TEMPERATURE (Deg C) (closest
dist above on X ...)
  Column 3: SST[X=@CDB:1] is SEA SURFACE TEMPERATURE (Deg C) (closest
dist below on X ...)

                SST     SST     SST
125W   / 108:  6.700   0.000   0.000
123W   / 109:   ....   8.000   2.000
121W   / 110:   ....   6.000   4.000
119W   / 111:   ....   4.000   6.000
117W   / 112:   ....   2.000   8.000
115W   / 113:  7.800   0.000   0.000
113W   / 114:  7.800   0.000   0.000
111W   / 115:   ....   2.000   2.000
109W   / 116:  8.300   0.000   0.000
```

## Ch3 Sec2.4.31. @CIA — closest index above

The transformation @CIA will compute at each grid point how far it is to the closest valid point above this coordinate position on the indicated axis. The distance will be reported in terms of the number of points (distance in index space). If a given grid point is valid (not missing) then the result of @CIA for that point will be 0.0. See the example for @CIB below. The units of the result are grid indices; integer number of grid units to the next valid point above. See General Information (p 76) for important details about this transformation, and see the example under @CIB below (p 91).

## Ch3 Sec2.4.32. @CIB — closest index below

The transformation @CIB will compute at each grid point how far it is to the closest valid point below this coordinate position on the indicated axis. The distance will be reported in terms of the number of points (distance in index space). If a given grid point is valid (not missing) then the result of @CIB for that point will be 0.0. The units of the result are grid indices, integer number of grid units to the next valid point below. See General Information (p 76) for important details about this transformation.

**Example:**

```
yes? USE coads_climatology
yes? SET REGION/x=125w:109w/y=55s/l=1
yes? LIST sst, sst[x=@cia], sst[x=@cib]   ! results below

  Column 1: SST is SEA SURFACE TEMPERATURE (Deg C)
  Column 2: SST[X=@CIA:1] is SEA SURFACE TEMPERATURE (Deg C) (closest
dist above on X ...)
  Column 3: SST[X=@CIB:1] is SEA SURFACE TEMPERATURE (Deg C) (closest
dist below on X ...)

                SST     SST     SST
125W   / 108:  6.700   0.000   0.000
123W   / 109:   ....   4.000   1.000
121W   / 110:   ....   3.000   2.000
119W   / 111:   ....   2.000   3.000
117W   / 112:   ....   1.000   4.000
115W   / 113:  7.800   0.000   0.000
113W   / 114:  7.800   0.000   0.000
111W   / 115:   ....   1.000   1.000
109W   / 116:  8.300   0.000   0.000
```

## Ch3 Sec2.5. IF-THEN logic ("masking")

Ferret expressions can contain embedded IF-THEN-ELSE logic. The syntax of the IF-THEN logic is simply (by example)

```
LET a = IF a1 GT b THEN a1 ELSE a2
```

(read as “if a1 is greater than b then a1 else a2”).

This syntax is especially useful in creating masks that can be used to perform calculations over regions of arbitrary shape. For example, we can compute the average air-sea temperature difference in regions of high wind speed using this logic:

```
SET DATA coads_climatology
SET REGION/X=100W:0/Y=0:80N/T=15-JAN
LET fast_wind = IF wspd GT 10 THEN 1
LET tdiff = airt - sst
LET fast_tdiff = tdiff * fast_wind
```

The user may find it clearer to think of this logic as WHERE-THEN-ELSE to aviod confusion with the IF used to control conditional execution of commands. Compound and multi-line IF-THEN-ELSE constructs are not allowed in embedded logic.

## Ch3 Sec2.6. Lists of constants (″constant arrays″)

The syntax {val1, val2, val3} is a quick way to enter a list of constants. For example

```
yes? LIST {1,3,5}, {1,,5}
  X: 0.5 to 3.5
  Column  1: {1,3,5}
  Column  2: {1,,5}
          {1,3,5} {1,,5}
 1   / 1:   1.000  1.000
 2   / 2:   3.000   ....
 3   / 3:   5.000  5.000
```

Note that a constant variable is always an array oriented in the X direction To create a constant aray oriented in, say, the Y direction use YSEQUENCE

```
yes? STAT/BRIEF YSEQUENCE({1,3,5})

  Total # of data points: 3 (1*3*1*1)
  # flagged as bad  data: 0
  Minimum value: 1
  Maximum value: 5
  Mean    value: 3 (unweighted average)
```

Below are two examples illustrating uses of constant arrays. (See the constant_array_demo journal file)

Ex. 1) plot a triangle (Figure 3_7)

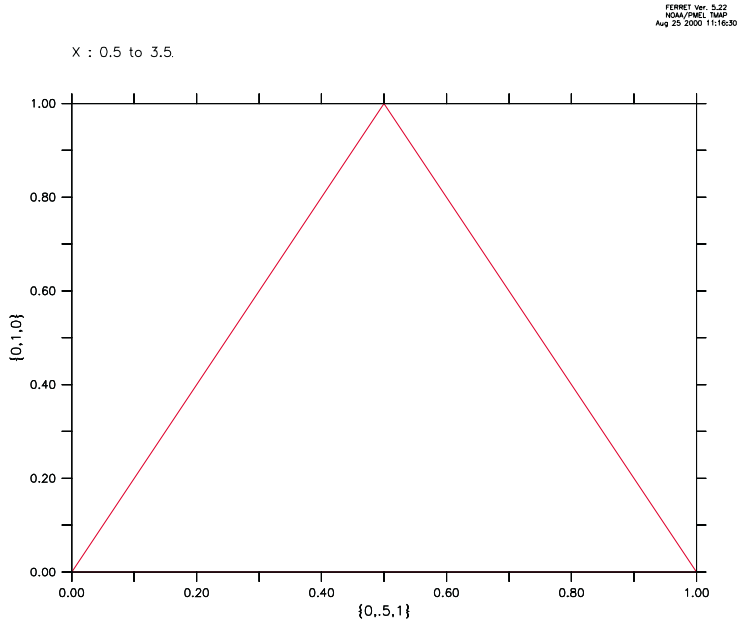
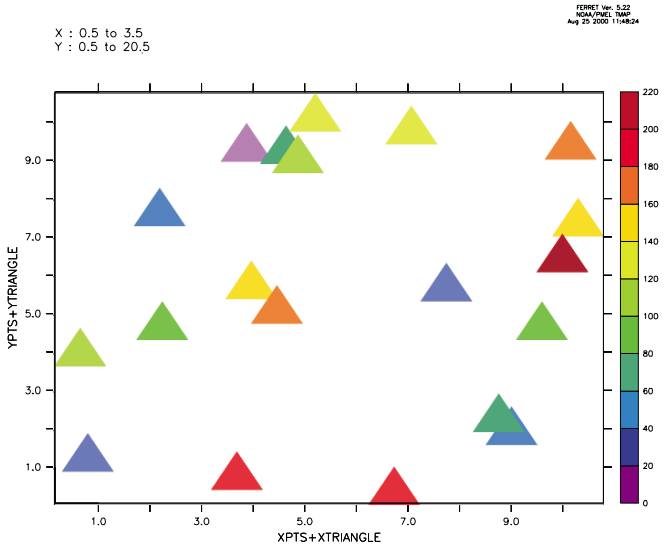



Figure 3_7

```
LET xtriangle = {0,.5,1}
LET ytriangle = {0,1,0}
POLYGON/LINE=8 xtriangle, ytriangle, 0
```

Or multiple triangles (Figure 3_8) See polymark.jnl regarding this figure




10* J[J=1:20]

Figure 3_8

Ex. 2) Sample Jan, June, and December from sst in coads_climatology

```
USE coads_climatology
LET my_sst_months = SAMPLEL({1,6,12}, sst)
STAT/BRIEF my_sst_months

yes? STAT/BRIEF my_sst_months

 Total # of data points: 48600 (180*90*1*3)
 # flagged as bad data: 21831
 Minimum value: -2.6
 Maximum value: 31.637
 Mean value: 17.571 (unweighted average)
```

## Ch3 Sec3. EMBEDDED EXPRESSIONS

Ferret supports “immediate mode” mathematical expressions—that is, numerical expressions that may be embedded anywhere within a command line. These expressions are evaluated immediately by Ferret—before the command itself is parsed and executed. Immediate mode expressions are enclosed in grave accents, the same syntax used by the Unix C shell. Prior to parsing and executing the command Ferret will replace the full grave accent expression, including the accent marks, with an ASCII string representing the numerical value. For example, if the given command is

```
CONTOUR/Z=`temp[X=180,Y=0,Z=@LOC:15]` salt
```

Ferret will evaluate the expression “temp[X=180,Y=0,Z=@LOC:15]” (the depth of the 15-degree isotherm at the equator/dateline—say, it is 234.5 meters). Ferret will generate and execute the command

```
CONTOUR/Z=234.5 salt
```

**Embedded expressions:**

Embedded expressions: the expression must evaluate to a single number, a scalar, or Ferret will respond that the command contains an error if the result is invalid the numerical string will be “bad” (see BAD= in following section, p. 96). Region qualifiers that begin a command containing an embedded expression will be used in the evaluation of the expression. If multiple embedded expressions are used in a single command they will be evaluated from left to right within the command. This means that embedded expressions used to specify region information (e.g., the above example) may influence the evaluation of other embedded expressions to the right. When embedded expressions are used within commands that are arguments of a REPEAT command their evaluation is deferred until the commands are actually executed. Thus the embedded expressions are re-evaluated at each loop index of the REPEAT command. Grave accents have a higher priority than any other syntax character. Thus grave accent expressions will be evaluated even if they are enclosed within quotation marks, parentheses, square brackets, etc. Substitutions based on dollar-signs (command script arguments and symbols) will be made before embedded expressions are evaluated. A double grave accent will be translated to a single grave accent and not actually evaluated. Thus double grave accents provide a mechanism to defer evaluation so that grave accent expressions may be passed to the Unix

command line with the SPAWN command or may be passed as arguments to GO scripts (to be evaluated INSIDE the script). The state of MODE VERIFY will determine if the evaluation of the embedded expression is echoed at the command line—similar to REPEAT loops.

## Ch3 Sec3.1. Special calculations using embedded expressions

By default Ferret formats the results of embedded expressions using 5 significant digits. If the result of the expression is invalid (e.g., 1/0) the result by default is the string "bad". Controls allow you to specify the formatting of embedded expression results in both valid and invalid cases and to query the size and shape of the result.

The syntax to achieve this control is KEYWORD=VALUE pairs inside the grave accents, following the expression and set off by commas. The recognized keywords are "BAD=", "PRECISION=", and "RETURN=". Only the first character of the keyword is significant, so they may be abbreviated as "B=", "P=", and "R=".

PRECISION=, BAD=, and RETURN= may be specified simultaneously, in any order, separated by commas. If RETURN= is included, however, the other keywords will be ignored.

**PRECISION=#digits**

can be used to control the number of significant digits displayed, up to a maximum of 10 (actually at most 7 digits are significant since Ferret calculations are performed in single precision). Ferret will, however, truncate terminating zeros following the decimal place. Thus

```
SAY `3/10,PRECISION=7`
```

will result in

```
0.3
```

instead of 0.3000000.

If the value specified for #digits is negative Ferret will interpret this as the desired number of decimal places rather than the number of significant digits. Thus

```
SAY `35501/100,P=-2`
```

will result in

```
355.01
```

instead of 355.

In the case of a negative precision value, Ferret will again drop terminating zeros to the right of the decimal point.

**BAD=string**

can be used to control the text which is produced when the result of the immediate mode expression is invalid. Thus

```
SAY `1/0,BAD=missing`
```

will result in

```
missing
```

or

```
SAY `1/0,B=-999`
```

will result in

```
-999
```

**RETURN=**

The keyword RETURN= can reveal the size and shape of the result. RETURN= may take arguments

- SHAPE
- ISTART, JSTART, KSTART, or LSTART
- IEND, JEND, KEND, or LEND
- XSTART, YSTART, ZSTART, or TSTART
- XEND, YEND, ZEND, or TEND
- SIZE
- T0
- UNIT
- TITLE
- GRID
- IAXIS, JAXIS, KAXIS, or LAXIS
- XAXIS, YAXIS, ZAXIS, or TAXIS

The RETURN= option in immediate mode expressions does not actually compute the result unless it must. For example, the expression

```
`sst, RETURN=TEND`
```

will return the formatted coordinate for the last point on the T axis of variable sst without actually reading or computing the values of sst. This allows Ferret scripts to be constructed so that they can anticipate the size of variables and act accordingly.

Note that this does not apply to variable definitions which involve grid-changing variables that return results on ABSTRACT axes. For those variables the size and shape of the result may depend on data values, so the entire result must be computed in order to determine many of the return= attributes

**RETURN=SHAPE**

returns the 4-dimensional shape of the result—i.e., a list of those axes along which the result comprises more than a single point. For example, a global sea surface temperature field at a single point in time:

```
SAY `SST[T=1-JAN-1983],RETURN=SHAPE`
```

will result in

```
XY
```

See Symbol Substitutions in the chapter “Handling String Data” (p. 168) for examples showing the special utility of this feature.

**RETURN=ISTART** (and similarly JSTART, KSTART, and LSTART)

returns the starting index of the result along the indicated axis: I, J, K, or L. For example, if CAST is a vertical profile with points every 10 meters of depth starting at 10 meters then Z=100 is the 10th vertical point, so

```
SAY `CAST[Z=100:200],RETURN=KSTART`
```

will result in

```
10
```

**RETURN=IEND** (and similarly JEND, KEND, and LEND)

returns the ending index of the result along the indicated axis: I, J, K, or L. In the example above

```
SAY `CAST[Z=100:200],RETURN=KEND`
```

will result in

```
20
```

The size and shape information revealed by RESULT= is useful in creating sophisticated scripts. For example, these lines could be used to verify that the user has passed a 1-dimensional field as the first argument to a script

```
LET my_expr = $1
DEFINE SYMBOL SHAPE `my_expr,RESULT=SHAPE`
QUERY/IGNORE ($SHAPE%|X|Y|Z|T|<Expression must be 1-dimensional%)
```

**RETURN=XSTART** (and similarly YSTART, ZSTART, and TSTART)

returns start of specified world coordinate region

**RETURN=XEND** (and similarly YEND, ZEND, and TEND)

returns end of specified world coordinate region

**RETURN=SIZE**

return the total number of points in the variable -- Nx*Ny*Nz*Nt

**RETURN=T0**

returns the T0 string from the time axis

**RETURN=UNIT**

returns the units string from the variable

**RETURN=TITLE**

returns the title of a variable

**RETURN=GRID**

returns the grid name of a variable

**RETURN=IAXIS** (and similarly JAXIS, KAXIS, and LAXIS)

Returns the name of an axis on which the variable is defined.

**RETURN=XAXIS** (and similarly YAXIS, ZAXIS, and TAXIS)

Returns the name of an axis on which the variable is defined.

## Ch3 Sec4. DEFINING NEW VARIABLES

The ability to define new variables lies at the heart of the computational power that Ferret provides. Complex analyses in Ferret generally proceed as hierarchies of simple variable definitions. As a simple example, suppose we wish to calculate the root mean squared value of variable, V, over 100 time steps. We could achieve this with the simple hierarchy of definitions:

```
LET v_rms     = v_mean_sq ^ 0.5
LET v_mean_sq = v_squared[L=@AVE]
LET v_squared = v * v
SET VARIABLE/TITLE="RMS V" v_rms

LIST/L=1:100 v_rms

(listed output not included)
```

As the example shows, the variables can be defined in any order and without knowledge in advance of the domain over which they will be evaluated. As variable definitions are given to Ferret with the LET (alias for DEFINE VARIABLE) command the expressions are parsed but not evaluated. Evaluation occurs only when an actual request for data is made. In the preceding example this is the point at which the LIST command is given. At that point Ferret uses the current context (SET REGION and SET DATA_SET) and the command qualifiers (e.g., "L=1:100") to determine the domain for evaluation. Ferret achieves great efficiency by evaluating only the minimum subset of data required to satisfy the request.

One consequence of this approach is that definitions such as

```
LET a = a + 1     ! nonsense
```

are nonsense within Ferret. The value(s) of variable "a" come into existence only as they are called for, thus it is nonsense for them to appear simultaneously on the left and right of an equal sign.

Variable names can be 1 to 24 characters in length and begin with a letter. See the command reference DEFINE VARIABLE (p. 260) for the available qualifiers.

## Ch3 Sec4.1. Global, local, and default variable definitions

All of the above definitions are examples of "global variable definitions." A global variable definition applies to all data sets. In the above example the expression "v_rms[D=dset_1]" would be based on the values and domain of the variable V from data set dset_1 and "v_rms[D=dset_2]" would similarly be drawn from data set dset_2. The domain of v_rms, its size, shape, and resolution, will depend on the particular data set in which it is evaluated.

Although global variables are simple to use they can lead to ambiguities. Suppose, for example, that data sets dset_1 and dset_2 contain the following variables:

```
Dset_1        dset_2
______        ______
speed          u, v
```

If we would like to compare speeds from the two data sets we might be tempted to define a new variable, speed, as

```
LET speed = (u*u + v*v)^0.5
```

In doing so, however, we create an ambiguity of interpretation for the expression "speed[d=dset_1]".

To avoid this ambiguity we need to create a variable definition, "speed," that is local to data set dset_2. The qualifier /D= used as follows

```
LET/D=dset_2 speed = (u*u + v*v)^0.5          ! in dset_2, only
```

provides this capability. The use of /D=dset_2 indicates that this new definition of "speed" applies only to data set dset_2.

A convenient shortcut is often to define a "default variable." A default variable is defined using the /D qualifier with no argument

```
LET/D speed = (u*u + v*v)^0.5  ! where "speed" doesn't already exist
```

As a default variable "speed" is a definition that applies only to data sets that would otherwise not posses a variable named speed. In this sense it is a fallback default.

## Ch3 Sec5. DEBUGGING COMPLEX HIERARCHIES OF EXPRESSIONS

A complex analysis generally proceeds within Ferret as a complex hierarchy of expressions: variables defined in terms of other variables defined in terms of other variables, etc., often containing many levels of transformation. When an error message such as "can only contour or vector a 2D region" occurs it may appear difficult to locate the reason for this message.

A simple strategy to locate the source of such problems is to use the command STAT which shows the size and shape of variables and expressions (simply edit the offending command line, replacing the PLOT, CONTOUR, VECTOR, etc. command with STAT and eliminating qualifiers if necessary) and use SHOW VARIABLE to see the variable definitions. By repeatedly using STAT to examine the component variables of definitions one can quickly locate the source of the problem.

# Chapter 4: GRIDS AND REGIONS

## Ch4 Sec1. OVERVIEW

Information describing a region in space/time, a data set, and a grid is collectively referred to as the "context." The current context may be examined with the commands SHOW DATA_SET, SHOW REGION, and SHOW GRID. The context may be set explicitly with the commands SET DATA_SET, SET REGION, and SET GRID.

The context may be modified for the duration of a single command with qualifiers to the command name (separated by slashes). The same qualifiers in square brackets may also modify single variables, changing the context only of that variable:

```
yes? PLOT/D=levitus_climatology  temp, salt
yes? CONTOUR  rose[D=etopo20]
yes? FILL/Z=0  temp[L=2] - temp[L=1]
```

## Ch4 Sec2. GRIDS

Every variable has an underlying grid which defines a coordinate space. All grids are in a sense 4 dimensional (X, Y, Z, and T) but axes normal to the data are represented as "normal" (such as the Z axis of the surface wind stress).

Grids can be viewed, specified and created using SHOW GRID, SET GRID, DEFINE AXIS, and DEFINE GRID. These commands are all in the Commands Reference section of this manual. Data can be regridded by the G= modifier. (See this chapter, section "Regridding," p. 107)

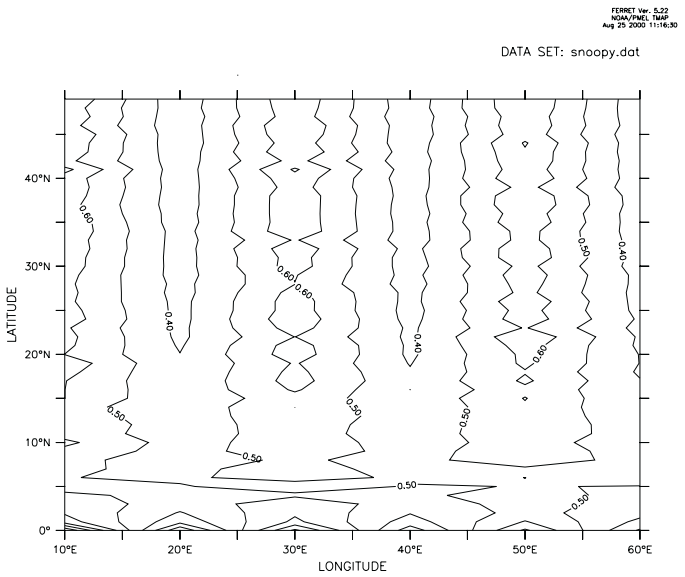
## Ch4 Sec2.1. Defining grids

Axes and grids can be explicitly created by DEFINE AXIS and DEFINE GRID. NetCDF and TMAP-formatted data set variables have all of the necessary grid and axis definitions embedded in the data set files, but if you are reading data from an ASCII or binary file, you must tell Ferret about the underlying grid of your data.

If you are creating abstract expressions entirely from pseudo-variables, you may want to define a grid in order to define the coordinate space of your calculation. This will also help produce a nicely labeled plot. (See the chapter "Variables and Expressions", "Grids and axes of pseudo-variables" (p. 48) and the example in the section on "Abstract Variables," p. 50.)

**Example**

This example is taken from the demonstration script “file_reading_demo.jnl”. An ASCII file contains a grid of numbers, 50 rows by 6 columns. Suppose the data are on a 2D grid of 6 longitudes by 50 latitudes (Figure 4_1).




MY_2D_VAR

Figure 4_1

```
yes? DEFINE AXIS/X=10E:60E:10/UNIT=DEGREE xlong
yes? DEFINE AXIS/Y=0:49N:1/UNIT=DEGREE ylat
yes? DEFINE GRID/X=xlong/Y=ylat  gsnoopy2d
! By default only 1 column is read, /COLUMNS= specifies 6 columns
yes? FILE/VAR=my_2D_var/COL=6/GRID=gsnoopy2d snoopy.dat
yes? CONTOUR my_2D_var
```

## Ch4 Sec2.2. Dynamic grids and axes

The commands DEFINE AXIS and DEFINE GRID, described in the preceding section, should be used when the grid or axis will be referenced more than once and/or shared among several variables. In many cases it is more convenient to use dynamic (a.k.a. “implicit”) grids and axes. Two quick examples:

```
PLOT SIN(X[X=0:3.14:.1])
```

– dynamically creates an axis from 0 to 3.14 by 0.1

```
SHADE SST[X=140E:160W:5, D=coads_climatology]
```

– dynamically creates a longitude axis extending from 140E to 160W by 5 degrees, dynamically creates a grid which is like the grid upon which the variable SST is defined but with the X axis replaced by the new dynamic axis, and automatically regrids to this new grid.

## Ch4 Sec2.2.1. Dynamic grids

It is often possible to avoid explicitly defining grids. This is useful in two common situations:

- **Situation 1**

  Regridding to specified axes without the need for defining the destination grid.

  Syntax: `G*=name@transform`, where

  | | |
  |---|---|
  | * | – The orientation of the axis to be regridded: "X," "Y," "Z," or "T" |
  | name | – The name of an axis or of another variable defined on the desired axis |
  | @transform | – The (optional) name of a regridding transform |

  **Example:**

  ```
  sst[GX=x10deg]
  ```

  Suppose the variable SST is defined on a 2×2 degree grid in latitude/longitude (e.g., SET DATA coads_climatology). If we wish to regrid to 10-degree spacing in longitude over a range from 175W to 75W we could use the commands

  ```
  DEFINE AXIS/X=175w:75w:10/UNITS=degrees x10deg
  LET sst10 = sst[GX=x10deg]
  ```

  Several axes can be specified together when they are to be regridded similarly. For example, instead of `sst[GX=x10deg, GY=x10deg]` one can use the more concise `sstGXY=x10deg]`

  Similarly, `av_sstGZ=@AVE, GT=@AVE]` can be condensed to `av_sst[GZT=@AVE]`

  Ferret will dynamically create a grid equivalent to new_grid in

  ```
  DEFINE GRID/LIKE=sst/X=x10deg new_grid.
  ```

Figure 4_2 shows the effects of regridding the 2×2 degree COADS data to a 10-degree spacing in longitude using (default) linear interpolation.

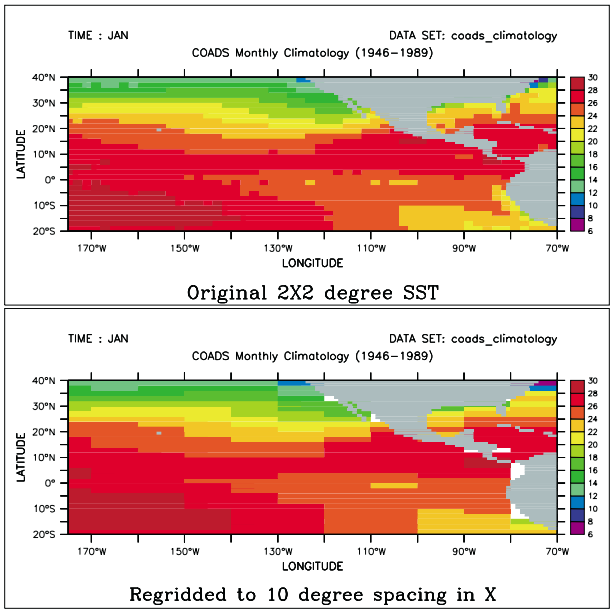



Figure 4_2

The command SHOW GRID SST10 will show the dynamically created grid. The names of dynamic grids and axes will always be displayed in parentheses.

Note that the transformation method to be used for regridding may also be specified, so LET SST10 = SST[GX=x10deg@ave] would create a 10-degree spaced result in which each grid point was computed as the weighted sum of the source points that fell within its grid box. The default method for regridding is linear interpolation.

- **Situation 2**

  Automatic reconciliation of incompatible grid shapes

  Syntax: `G=name@transform`

  where

  | | |
  |---|---|
  | name | – The name of a grid or of another variable defined on the desired grid |
  | @transform | – The (optional) name of a regridding transform |

  **Example:**

  ```
  VAR1[g=VAR2]
  ```

  If two variables are defined on grids that are mutually non-conformable because axes exist in one grid but do not exist (are NORMAL) in another, Ferret will now create a dynamic grid to resolve the non-conformabilities. This means that an expression of the form VAR1[G=VAR2] will be meaningful as long as the grid domains overlap.

  For example, TEMP[d=levitus_climatology] is defined on an XYZ (time-independent) grid whereas SST[d=coads_climatology] is defined on an XYT grid. So to evaluate the expression SST[d=coads_climatology,G=TEMP[d=levitus_climatology]] Ferret will create a dynamic intermediate grid equivalent to

  ```
  DEFINE GRID/LIKE=sst[D=coads_climatology]/X=temp/Y=temp
  ```

  so that regridding occurs on the X and Y axes but the original grid structure is maintained with respect to depth and time.

  The command SHOW GRID will reveal the resulting dynamically created grid structure.

## Ch4 Sec2.2.2. Dynamic axes

The syntax "GX=lo:hi:delta" can be used in square brackets modifying a variable name to indicate the dynamic creation of an axis with the indicated range and spacing and the immediate regridding of the variable to a grid containing that axis. For example, SST[GX=175W:75W:10] will create a dynamic axis of 10-degree regular point spacing, will create a dynamic grid incorporating this axis (see previous section), and will regrid the variable SST to this grid.

Similarly, by referring to the grid indices rather than their world coordinates, the expression SST[GI=1:100:5] will create a dynamic axis that subsamples every 5th longitude point from

SST. In this case the points of the resulting axis may be irregularly spaced if the points of the original axis were also irregular.

As with the dynamic regridding described above, transformations can be specified to indicate the regridding technique. Thus SST[GI=1:100:5@AVE] will use averaging instead of the default linear interpolation to perform the regridding.

As a notational convenience the "G" may be dropped when referring to dynamic axes. Thus SST[X=175W:75W:10] is equivalent to SST[GX=175W:75W:10] and SST[I=1:100:5@AVE] is equivalent to SST[GI=1:100:5@AVE]. When using this notational convenience keep in mind that a regridding is taking place, so the transformation applied (if any) must be a regridding transformation (see SHOW TRANSFORMS in the command reference section, p. 325).

The lower plot of Figure 4_2 illustrates the effect of dynamic axes in the command

```
SHADE SST[GX=175W:75W:10]
```

## Ch4 Sec2.2.3. Dynamic pseudo-variables

The same notation used for dynamic axes may also be applied to pseudo-variables providing a simple means for creating arrays of values. For example, X[GX=0.2:1:0.2] is a vector of 5 points from 0.2 to 1 at a regular spacing of 0.2 units. The vector is oriented in the X direction.

An example of using such a vector is (Figure 4_3)

```
PLOT SIN(X[GX=0:3.14:.1])
```

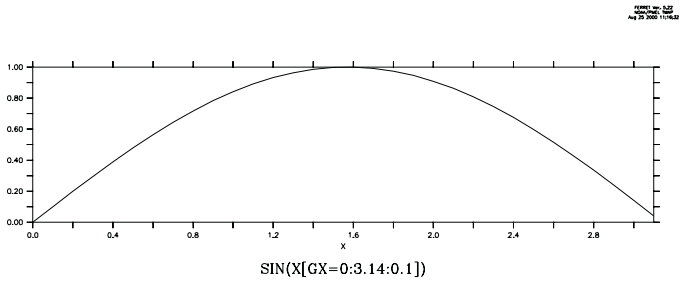



Figure 4_3

Note that when the lo:high:delta notation is applied to T or L expressed as calendar dates the units of the delta value will be **hours.**For example, L[GT=1-jan-1980:1-feb-1980:24] is the integers 1 to 32 defined on an axis of 32 days, 24 hours apart.

As a notational convenience the "G" may be dropped when referring to dynamic pseudo-variables. Thus X[X=0.2:1:0.2] is equivalent to X[GX=0.2:1:0.2].

See also the discussion of grids for pseudo-variables in section 3.1.3, p. 48.

## Ch4 Sec2.3. Regridding

Syntax:

`var[G=name]` for (default) linear interpolation to new grid

or

`var[G=name@trn]` to regrid all axes using transform "trn" (see below)

or

`var[G=name,GX=@TRN,GY=@TRN,...]` to control regridding transformations along each axis separately

where

`var` is the name of the variable to be regridded (e.g., temp, u, tau, ...)

`name` is the name of a variable (e.g., temp[G=u]) or the name of a grid (e.g., temp[G=gu01])

`trn` is the name of a special transformation (e.g., @AVE, @ASN, @LIN)

The syntax `var[G=name,GX=@TRN,GY=@TRN,...]` can be compressed when specifying that several axes are to be regridded similarly. For example, instead of

```
var[GX=sst, GY=SST]
```

one can now use the more concise

```
var[GXY=sst]
```

Similarly, if using a regridding transformation,

```
var[GZ=@AVE, GT=@AVE]
```

can be condensed to

```
var[GZT=@AVE]
```

Note that in Ferret Version 5 and after when the limits of a variable are unspecified v2[g=v1] will default to the full extent of the v1 grid. Previously, it would become the size of whatever region of the v2 native grid overlapped with the v1 grid.

The Ferret distribution provides a demonstration of many regridding techniques:

```
yes? GO regridding_demo
```

Regridding is essential for algebraic operations that combine variables on incompatible grids. Ferret provides the commands DEFINE AXIS and DEFINE GRID to assist with the creation of arbitrary grids.

The result grid of a regridding operation does not necessarily match exactly the destination grid requested. For example, suppose the native grid of variable TEMP3D (Ocean Temperature) is 1 degree resolution in X and Y and 50 meter spacing in Z. If the syntax "[G=sst]" is used to request regridding to the grid of variable SST (Sea Surface Temperature), which is 2 degree reso-

lution in X and Y, but normal to Z, then the resulting grid will be generated dynamically—inheriting X and Y axes from SST as requested, but retaining the Z axis of TEMP3D.

**Examples**

1) Suppose the variables u and temp are on staggered X, Y, and Z axes but share the same T axis. Then the two variables can be multiplied together on the axes (grid) of the u variable as follows:

```
yes? CONTOUR u * temp[G=u]
```

   This will regrid temp onto the u grid by multi-axis linear interpolation before performing the multiplication.

2) Two variables, v1 and v2, are defined on distinct 4-dimensional grids (X, Y, Z, and T axes). The T axes of the two grids are identical but the X, Y, and Z axes all differ between the two variables. (This is often the case in numerical model outputs.)

   To obtain the variable v1 on its original Z (depth) locations but regridded in the XY plane to the grid locations of the variable v2, define a new grid (say, named "new_grid") that has the X and Y axes of v2 but the Z axis of v1.

```
yes? DEFINE GRID/LIKE=v2/Z=v1 new_grid        !define new grid
yes? LIST/X=160E:140W/Y=5S:5N v1[G=new_grid]  !request regridding
```

3) In this example we look at temperature data from two data sets. levitus_climatology, an annual climatology, has the variable "temp" on an XYZ grid which is 1×1 degree in XY, and coads_climatology, a monthly climatology, has the variable "sst" on an XYT grid which is 2×2 degrees in XY. Suppose we wish to look at the sea surface temperatures in January at the higher XY resolution of the Levitus data.

```
yes? SET DATA levitus_climatology
yes? SET DATA coads_climatology
yes? SET REGION/L=1/Z=0
yes? !get the name of the grid on which temp is defined
yes? SHOW GRID temp[D=levitus_climatology]  ! -> "Glevitr1"
yes? DEFINE GRID/X=glevitr1/Y=glevitr1/Z=sst/L=sst glevitus_xy
yes? LIST/X=150E:155E/Y=0:5N sst[G=glevitus_xy]
```

## Ch4 Sec2.3.1. Regridding transformations

Ferret supports several regridding transformations. Use the SHOW TRANSFORMATIONS command to obtain a list of the supported transformations from Ferret. The choice of regridding transformation determines the computation by which data from the source grid determine the values on the destination grid.

Regridding transformations provide a means to perform a given calculation over a limited span of coordinates and repeat that calculation for a series of contiguous spans. For example, if we wish to compute the variance of the variable SST over 10-degree longitude range from 180 to 170W we could use the syntax `sst[X=180:170w@VAR]`. Now, if we wish to perform the same operation 10 times in 10-degree wide bands from 180 to 80W we could instead use `G=@VAR` regridding as in (see Dynamic Grids, p. 103, for an explanation of the "GX=" syntax):

```
DEFINE AXIS/X=175w:85w:10/UNITS=degrees ×10deg
LET sst10 = sst[GX=x10deg@VAR]
```

The regridding transformations are:

@LIN—linear interpolation (the default if no transform is specified)
Performs regridding by multi-axis linear interpolation.

@AVE—averaging
Computes the length-weighted average of all points on the source grid that lie partly or completely within each grid cell of the destination grid.

**Note:** When @AVE is applied simultaneously to the X and Y axes, where X and Y are longitude and latitude, respectively, an area-weighted average (weighted by cos(latitude)) is used. The @AVE transformation is unique in this respect. In multiple axis applications other than X and Y @AVE will be applied sequentially to the axes, computing the "average of the average." This may not be the desired weighting scheme in some cases. See @VAR below for an example.

@ASN—(blind) association
Associates by subscript (blindly) the points from the source grid onto destination coordinates.

@VAR
Computes the variance of the points from the source grid that fall within each destination grid cell. This is a length-weighted computation like the @AVE transformation.

**Note:** This transformation is suitable for regridding only in a single axis. When applied simultaneously to two axes, for example, it will compute the variance of the variance. For example, V[gx=130E:80W:10@VAR, gy=205:20W:10@VAR] is equivalent to tmp[X=130E:80W:10@VAR] where tmp=V[y=20S:20N:10@VAR].

@NGD
Compute the number of points from the source grid that fall within each destination grid cell. Note that the results of this calculation need not be integers—this is a length-weighted computation like the @AVE transformation. It is common for a grid cell on the source grid to span the boundary between grid cells on the destination grid, thereby contributing a fraction of its weight to multiple destination grid cells.

**Note:** This transformation is suitable only for regridding on a single axis. When applied simultaneously to two axes, for example, it will compute a constant. See @VAR for an example.

@SUM

Computes the length-weighted sum of the points from the source grid that fall within each destination grid cell. This is a length-weighted computation like the @AVE transformation.

@MIN

Finds the minimum value of those points from the source grid that lie within each destination grid cell. Note that this is NOT a weighted calculation; the destination grid cell that "owns" a source point is determined entirely from the coordinate location of the source point, not from the limits of the source grid cell.

(As of Ferret V5.1) If a point on the source axis lies exactly on the boundary between grid cells on the destination axis it will be included in the calculations for the higher indexed cell on the destination axis. If a point on the source axis lies exactly on the upper cell boundary of the highest point on the destination axis, then it will be included in the calculations for that highest destination grid cell.

If you have data on a single point axis and you wish to embed it in a larger axis range you can achieve this by using either the G=@MIN or G=@MAX. For example,

```
yes? define axis/x=163e/npoints=1 x1pt
yes? let var_1pt = randu(x[gx=x1pt])    ! a random value at a single
coordinate
yes? list var_1pt
            RANDU(X[GX=X1PT])
            LONGITUDE: 163E
       0.4914
yes? define axis/x=161e:165e:1 x5pt
yes? list var_1pt[gx=x5pt@max]     ! same value embedded within 5 point
axis
            RANDU(X[GX=X1PT])
             regrid: 1 deg on X@MAX
 161E   / 1:    ....
 162E   / 2:    ....
 163E   / 3:  0.4914
 164E   / 4:    ....
 165E   / 5:    ....
```

@MAX

Finds the maximum value of those points from the source grid that lie within each destination grid cell. Note that this is NOT a weighted calculation; the destination grid cell that "owns" a source point is determined entirely from the coordinate location of the source point, not from the limits of the source grid cell..

(As of Ferret V5.1) If a point on the source axis lies exactly on the boundary between grid cells on the destination axis it will be included in the calculations for the higher indexed cell

on the destination axis. If a point on the source axis lies exactly on the upper cell boundary of the highest point on the destination axis, then it will be included in the calculations for that highest destination grid cell.

The @MAX transformation is useful as a mechanism to perform regridding between two axes that do not quite match. A common example would be to regrid between two monthly axes one of which has points located at the 15th of each month and the other having points exactly at mid-month. These Ferret commands illustrate the example using a 5-month axis in 1993:

```
! define axes for 15th of month and mid-mont
yes? DEFINE AXIS/UNIT=DAYS/T0=1-JAN-1900 month_15 = DAYS1900(1993,I[I1:5],
15)

yes? DEFINE AXIS/UNIT=DAYS/T0=1-JAN-1900/EDGES month_mid =
    DAYS1900(1993,I[I=1:6], 1)
yes? let my_var = L[gt=month_15

yes? list my_var
               L[GT=MONTH_15]

      15-JAN-1993 00 / 1:  1.000
      15-FEB-1993 00 / 2:  2.000
      15-MAR-1993 00 / 3:  3.000
      15-APR-1993 00 / 4:  4.000
      15-MAY-1993 00 / 5:  5.000


yes? list my_var[gt=month_mid]

               L[GT=MONTH_15]

               regrid: on T

      16-JAN-1993 12 / 1:  1.048
      15-FEB-1993 00 / 2:  2.000
      16-MAR-1993 12 / 3:  3.048
      16-APR-1993 00 / 4:  4.033
     16-MAY-1993 12 / 5:   ....    ! unable to interpolate

yes? list my_var[gt=month_mid@max]

               L[GT=MONTH_15]
```

```
            regrid: on T@MAX

 16-JAN-1993 12 / 1:  1.000
 15-FEB-1993 00 / 2:  2.000
 16-MAR-1993 12 / 3:  3.000
 16-APR-1993 00 / 4:  4.000
16-MAY-1993 12 / 5:  5.000
```

@XACT

Regridding with G=@XACT (or GX=@XACT, etc.) is a request to transfer values from a source grid to a destination grid only at those positions where there is an exact coordinate match between the source and destination axis points on the axis in question. Other destination points will be set to "missing". This transformation is especially useful for taking multiple in-situ data profiles, such as oceanographic cast data, and regridding them onto a regular (sparse) grid. For example: grep

```
yes? LET xcoarse = sin(x[x=0:20:10])
yes? LIST xcoarse
               SIN(X[X=0:20:10])
          0  / 1:  0.0000
          10 / 2: -0.5440
         20 / 3:  0.9129
    yes? DEFINE AXIS/X=0:20:5 xfine
    yes? LIST xcoarse[gx=xfine@XACT]
               SIN(X[X=0:20:10])
               regrid: 5 delta on X@XACT
        0    / 1:    0.0000
        5    / 2:      ....
        10   / 3:  -0.5440
        15   / 4:     ....
        20  / 5:    0.9129
```

@MOD

Creates climatologies from time series by regridding to a time series axis with a modulo regridding transformation. See section Modulo Regridding (p. 113) for details.

**Examples**

1) Let variable temp be defined on a grid with points spaced regularly at 1-degree intervals in both longitude and latitude (X and Y). Let grid "g10" possess points spaced regularly at 10-degree intervals in both X and Y.

```
yes? PLOT temp[G=g10]           ! uses linear interpolation (@LIN)
yes? PLOT temp[G=g10@AVE]       ! area-averages 10x10 degrees of
source\
                                  points into each destination point.
yes? PLOT temp[G=g10,GX=@AVE]   ! averages 10 degrees of longitude but\
                                  interpolates (@LIN) in Y.
```

2) @ASN has the effect of bypassing Ferret's protections against misrepresenting data (Figure 4_4).

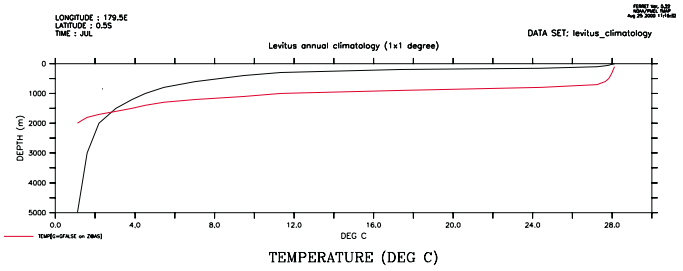



Figure 4_4

```
yes? SET DATA levitus_climatology
yes? SET REGION/X=180/Y=0     ! true profile
yes? PLOT/Z=0:5000 temp
yes? DEFINE AXIS/DEPTH /Z=100:2000:100  zfalse
yes? DEFINE GRID/LIKE=temp /Z=zfalse  gfalse   ! false profile
yes? PLOT/Z=0:5000/OVER temp[G=gfalse@ASN]
```

## Ch4 Sec2.4. Modulo regridding

Ferret can create climatologies from time series simply by regridding to a climatological axis with a modulo regridding transformation. For example, if the axis named month_reg is a 12-point monthly climatological (modulo) axis then the expression

```
LET sst_climatology = sst[D=coads,GT=month_reg@MOD]
```

is a 12-month climatology computed by averaging the full time domain of the input variable (576 points for coads) modulo fashion into the 12 points of the climatological axis.

Ferret has three pre-defined climatological axes: seasonal_reg (Feb, May, Aug, Nov), month_reg (middle of every month), and month_irreg (15th of every month).

```
yes? USE climatological_axes
*** NOTE: regarding ... climatological_axes.cdf
*** NOTE: Climatological axes SEASONAL_REG, MONTH_REG, and MONTH_IRREG
    defined
yes? CANCEL DATA climatological_axes ! the axes are still defined
```

To generate a climatology based on a restricted range of input data simply define an intermediate variable with the desired points. For example, a monthly climatological time series based on data from the 1960s could be computed using

```
LET sst_1960s = sst[D=coads,T=1-jan-1960:31-dec-1969]
PLOT sst_1960s[GT=month_reg@MOD]
```

In a similar fashion intermediate variables can be defined that mask out certain input points.

This example shows the entire sequence necessary to generate a plot of climatological SST at 40N, 40W based on the January 1982 to December 1992 Fleet Numerical wind data set. (Figure 4_5).

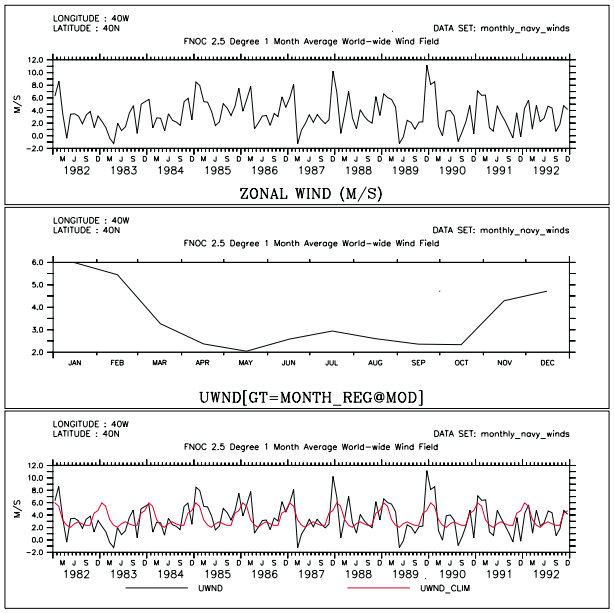



Figure 4_5

```
! use the predefined climatological axes
USE climatological_axes
CANCEL DATA climatological_axes

! use the Fleet Numerical winds
SET DATA monthly_navy_winds

! plot the raw data (top figure)
SET REGION/X=40w/Y=40n
plot uwnd

! plot the 12 month climatology (middle figure)
LET uwnd_clim = uwnd[GT=month_reg@MOD]
PLOT uwnd_clim

! since uwnd_clim is on a climatological axis
! Ferret can also plot it on the calendar axis with the raw data
PLOT/T=16-jan-1982:17-dec-1992 uwnd,uwnd_clim
```

In many cases the volume of input data needed to perform climatological calculations is very large. In the example above the command

```
CONTOUR/X=0:360/Y=90s:90n sst_climatology[L=1]
```

to plot January from the climatology would require Nx*Ny*Nt=72*72*576=3 Megawords of data. Such calculations may be too large to fit into memory. However, if the region is fully specified (as shown for the X and Y limits in the example) Ferret's internal memory manager will break up the calculation as needed to produce the result. (See Memory Use in the chapter "Computing Environment", p. 187, for further details.)

Unlike other transformations and regridding, modulo regridding is performed as an unweighted average: each non-missing source point contributes 100% of its weight to the destination grid box within which it falls. If the source and destination axes are not properly aligned this can lead to apparent shifts in the data. For example, if a monthly time series has data points at the first of each month and a climatological axis is defined at midmonths, then unweighted modulo averaging will lead to an apparent 1/2-month shift. To avoid situations of this type simply regrid to the climatological axis using linear interpolation prior to the modulo regridding.

Here is an example that illustrates the situation and the use of linear interpolation to repair it. (Figure 4_6).

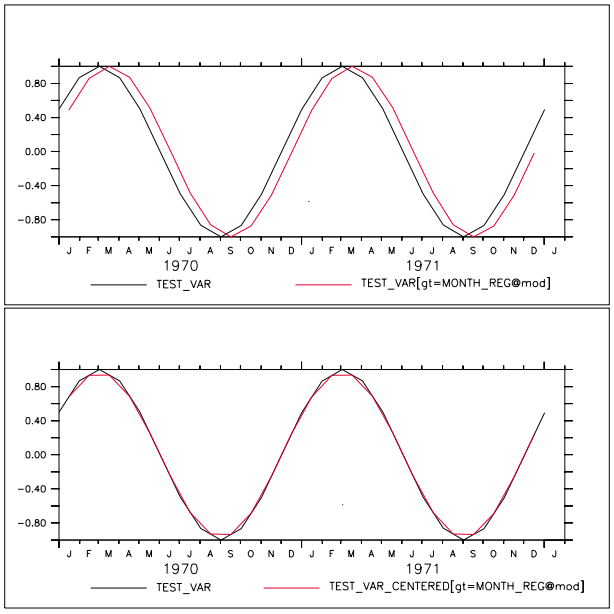



Figure 4_6

```
! define test_var, an illustrative variable with 1 year periodicity
! Note: test_var points are at the **beginnings** of months
DEFINE AXIS/T=1-jan-1970:1-jan-1974:`365.25/12`/UNITS=days t10years
DEFINE GRID/T=t10years gg
LET test_var = SIN(L[G=gg]*2*3.14/12)

! plot 4 years of the cycle
PLOT test_var
```

```
! define climatological axes at the midpoints of months
USE climatological_axes
CANC DATA climatological_axes

! notice that modulo regridding appears to shift the data
! (due to mis-aligned source and destination axes) (top figure)
PLOT/OVER/T=1-jan-1970:1-jan-1974 test_var[GT=month_reg@MOD]

! to avoid the shift we can first regrid test_var to mid-month
! points using linear interpolation (the default regridding method)
! notice that the function test_var is largely unchanged by this
LET test_var_centered = test_var[GT=month_reg]
PLOT/OVER/T=1-jan-1970:1-jan-1974 test_var_centered

! finally perform a modulo regridding on well-aligned data
! notice that the shift is gone (bottom figure)
PLOT/OVER/T=1-jan-1970:1-jan-1974 test_var_centered[GT=month_reg]
```

## Ch4 Sec2.4.1. Modulo regridding statistics

In addition to the modulo averaging calculation performed by @MOD Ferret provides other statistics of the regridding. All modulo regridding calculations are unweighted as discussed under @MOD.

@MODVAR
the variance of source points within each destination grid box (SUM(var-varbar)^2)/(n-1))

@MODSUM
the sum of the source points within each destination grid box

@MODNGD
the number of source points contributing to each destination grid box

@MODMIN
the minimum value of the source points contributing to each destination grid box

@MODMAX
the maximum value of the source points contributing to each destination grid box

## Ch4 Sec3. REGIONS

The region in space and time where expressions are evaluated may be specified in 3 different ways:

1) with the command SET REGION
2) with qualifiers to the command name (slash-delimited)
3) with qualifiers to variable names (in square brackets, comma-delimited)

If SET REGION is used, Ferret remembers the region as the default context for future commands, whereas a qualifier to a command name specifies the region for that command only, and a qualifier to a variable name specifies the region for that variable and command only.

Regions may be manipulated using DEFINE REGION, SET REGION, @ notation, and CANCEL REGION. The Commands Reference section of this manual covers all of these topics.

Region information is normally specified in the following form:

`QUAL=val` or
`QUAL=lo_val:hi_val` or
`QUAL=val@transform` (as a variable qualifier only) or
`QUAL=lo_val:hi_val@transform` (as a variable qualifier only)

When the region for an axis is specified as a single value (instead of a range) Ferret, by default, selects the grid point of the grid box containing this value. The Ferret mode "interpolate" can control this behavior. See command SET MODE INTERPOLATE in Commands Reference, p. 303.

**Examples: Regions**

Examples of valid region specifications.

1) Fully specify the region in an XY plane with the first vertical (Z) level and time 27739.

```
yes? SET REGION/X=140E:160W/Y=10S:20N/K=1/T=27739
```

2) Contour vertical heat advection within whatever region is the current default (previously set with SET REGION).

```
yes? CONTOUR qadz
```

3) Define, modify and set a named region and then modify with delta notation.

```
yes? DEFINE/REGION/Y=5S:5N YT          !define region YT to be 5S:5N
yes? DEFINE REGION/DY=-1:+1 YT         !modify region YT to be 6S:6N
yes? SET REGION/@YT                    !set current region to YT
yes? SET REGION/DY=-1:+1               !modify current region to 7S:7N
```

4) List meridional currents calculated by averaging values between the surface and a depth of 50 m.

```
yes? LIST v[Z=0:50@AVE]
```

5) Equivalent to `v[Z=10] - v[Z=0:100@AVE]`, the anomaly at z=10 between v and the 0 to 100 meter depth average of v.

```
yes? LIST/Z=10 v - v[Z=0:100@AVE]
```

## Ch4 Sec3.1. Latitude

Specify latitude or a latitude range with the qualifier Y or J. Specifications using J are integers between 1 and the number of points on the Y axis. Specifications using Y are in the units of the Y axis.

The units may be examined with SHOW GRID/Y. If the Y axis units are degrees of latitude then the Y positions may be specified as numbers followed by the letters “N” or “S”.

**Examples**

```
yes? CONTOUR temp[Y=15S:10N]
yes? LIST/J=50 u
```

## Ch4 Sec3.2. Longitude

Specify longitude or a longitude range with the qualifier X or I. Specifications using I are integers between 1 and the number of points on the X axis. Specifications using X are in the units of the X axis.

The units may be examined with SHOW GRID/X. If the units are degrees, then X values may be given as numbers followed by “W” or “E” (e.g., 160E, 110.5W) or as values between 0 and 360 with Greenwich at 0 increasing eastward. **Note:** If the X axis is “modulo” then it is sometimes desirable to use X greater than 360.

**Examples**

```
yes? CONTOUR  temp[Y=160E:140W]
yes? LIST/I=100  u
yes? SHADE/X=100:460  temp        !360 degrees centered at 100W
```

See the chapter “Grids and Regins”, section “Modulo Axes” (p. 121), for help with globe-encircling axes.

## Ch4 Sec3.3. Depth

Specify depth or a depth range with the qualifier Z or K. Specifications using K are integers between 1 and the number of points on the Z axis. Specifications using Z are in the units of the Z axis.

The units may be examined with SHOW GRID/Z.

**Examples**

```
yes? CONTOUR temp[Z=0:100]
yes? LIST/K=3 u
```

## Ch4 Sec3.4. Time

Specify time or a time range with the qualifier T or L. Specifications using L are integers between 1 and the number of points on the T axis. Specifications using T may refer to calendar dates or to the time step units in which the time axis of the data set is defined.

Calendar date/time values are entered in the format dd-mmm-yyyy:hh:mm:ss, for example 14-FEB-1988:12:30:00. At a minimum the string must contain day, month, and year. If the string contains any colons it must be enclosed in quotation marks to differentiate from colons used to designate a range. If a time increment is specified with the repeat command given in calendar format (e.g., REPEAT/T="1-JAN-1982":"15-JAN-1982":6) it is interpreted as hours always. Calendar dates in the years 0000 and 0001 are regarded as year-independent dates (suitable for climatological data).

SHOW GRID/T can be used to display time step values. (Units may vary between data sets.) The commands SET MODE CALENDAR and CANCEL MODE CALENDAR can be used to view date strings or time steps, respectively.

**Examples**

```
yes? LIST/T="1-JAN-1982:13:50":"15-FEB-1982"   density
yes? CONTOUR temp[T=27740:30000]
yes? LIST/L=90 u
```

See the section in this chapter on “Modulo Axes” (p. 121) for help with climatological axes.

## Ch4 Sec3.5. Delta

The notation q=lo:hi:delta (e.g., Y=20S:20N:5) specifies that the data in the requested range is regularly subsampled at interval “delta.”

This notation is valid only for the REPEAT, SHOW GRID, and DEFINE AXIS commands, and the qualifiers /XLIMITS and /YLIMITS used in action commands with graphical output.

### @ notation

Regions may be named and referred to using the syntax “@name”. Some commonly used regions are predefined. See commands SET REGION (p. 309) and DEFINE REGION (p. 259) in the Commands Reference section for further information.

If a region is specified using a combination of "@" notation and explicit axis limits the explicit axis limits will be evaluated after the "@" specification, possibly superseding the "@" limits.

**Note:** It is not advised to use the @notation inside of variable definitions, as redefinitions of the named region can cause code errors that lead to wrong results.

Using the @ notation only sets/alters the axis limits specified in the named region. For example, suppose that region "XY" is defined for the X and Y axes, but not for the Z and T axes. Then

```
yes? SET REGION/@XY
```

modifies only X and Y limits. BUT,

```
yes? SET REGION XY
```

modifies all axes—X and Y to the limits specified by XY, and Z and T to unspecified (even if they were previously specified).

**Examples**

1) Contour the 25th time step of temperature data at depth 10 within region T, the "Tropical Pacific."

```
yes? CONTOUR/@T/Z=10/L=25 temp
```

2) Produce a contour plot over region W, the "Whole Pacific Ocean," in the XY plane (the variable to be contoured as well as the depth and time will be inferred from the current context).

```
yes? CONTOUR/@W var1
```

3) Set the default region to "T", the Tropical Pacific Ocean (latitude 23.5S to 23.5N).

```
yes? SET REGION/@T
```

4) Define a region and then supersede with an axis limit specification.

```
yes? DEFINE REGION/X=180:140W/Y=2S:2N/Z=5   BOX1
yes? SET REGION/@BOX1/Z=15                  !replace Z
```

**Pre-defined regions**

As a convenience in the analysis of the Tropical Pacific Ocean the following regions are pre-defined:

| Name | Region | Latitude | Longitude |
|---|---|---|---|
| T | Tropical Pacific | 23.5S:23.5N | 130E:70W |
| N | Narrow Pacific | 10.0S:10.0N | 130E:70W |
| W | Whole Pacific | 30.0S:50.0N | 130E:70W |

These may be redefined by the user for the duration of a Ferret session or until the definitions are canceled.

## Ch4 Sec3.6. Modulo axes

Some axes are inherently "modulo," indicating that the axis wraps around—the first point immediately following the last.

To determine if an axis is modulo use SHOW AXIS or SHOW GRID. A letter "m" following the number of points in the axis indicates a modulo axis. The command SHOW GRID qualified by the appropriate axis limits can be used to examine any part of the axis—including points beyond the nominal length of the axis. The commands SET AXIS/MODULO and CANCEL AXIS/MODULO can be used to control this feature on an axis-by-axis basis.

**Example**

```
yes? SET DATA coads_climatology
yes? SHOW GRID/I=180:183 sst  !range request beyond last point
  GRID COADS1
  name       axis       # pts     start            end
  COADSX  LONGITUDE     180mr     21E              19E(379)
  [text omitted]
        I     X                BOX_SIZ
      180>  19E(379)              2
      181>  21E(381)              2
      182>  23E(383)              2
      183>  25E(385)              2
```

The most common uses of modulo axes are:

1) As longitude axes for globe-encircling data sets. This allows any starting and any ending longitudes to be used, for example, X=140E:140E indicates the entire earth with data beginning and ending at 140E.
2) As time axes for climatological data. By this device the time axis appears to extend from 0 to infinity and the climatological data can be referred to at any point in time. This facilitates comparisons with data sets at fixed times.

## Ch4 Sec3.7. Region Conflicts

Conflicting region information can be given to Ferret in obvious ways such as

```
LIST/I=1:3 I[I=1:10]
```

in which it is not clear if the request is for 10 points or for 3, or in subtler, disguised ways such as

```
LET A = I[I=1:10] LIST/I=1:3 A
```

In both examples Ferret would resolve the conflict by listing just the three values I=1:3.

Internally, Ferret uses the region **closest to the variable** to perform the calculation. Thus, in both of the examples above Ferret will perform the calculation on I=1:10, since the "[I=1:10]" directly modifies the variable name. If Ferret sees conflicting regions it attempts to use the regions further from the variable to clip the calculation. Thus 10 points are clipped to 3 in the above examples.

Unresolvable conflicts such as

```
LIST/I=11:13 I[I=1:10]
```

will result in a warning message that invalid limits have been ignored.

# Chapter 5: ANIMATIONS AND GIF IMAGES

## Ch5 Sec1. OVERVIEW

A sequence of Ferret plots can be stored and then animated. Each plot is stored as one frame in a movie file. Ferret stores movie frames in Hierarchical Data Format (HDF), a format designed by the National Center for Supercomputing Applications (NCSA). A movie file can then be displayed as an animated sequence of frames with NCSA's xds—X Data Slice (not distributed with Ferret; see the section in this chapter "Displaying an HDF movie" (p. 124), for details).

## Ch5 Sec2. CREATING AN HDF MOVIE

Creating a movie requires two steps:

1) designate an output file with SET MOVIE
2) generate a sequence of frames with REPEAT and FRAME

See commands SET MOVIE (p. 308), CANCEL MOVIE (p. 245), SHOW MOVIE (p. 324), FRAME (p. 265), and REPEAT (p. 286) in the Commands Reference section of this manual.

**Example: basic movie**

```
yes? SET DATA coads_climatology       !specify data set
yes? SET REGION/@W                    !specify Pacific Ocean
yes? LET/TITLE="SST Anomaly" SST_ANOM = SST - SST[L=1:12@AVE]
yes? REPEAT/L=1:12 (FILL sst_anom; FRAME/FILE=my_movie.mgm)
                                      !filled contour of sea surface\
                                      temp anomaly captured and\
                                      written to HDF file
```

Optionally, ".mgm" will be assigned to the movie file.

REPEAT executes its argument (in the above example, FILL) successively for each timestep specified. REPEAT can have multiple arguments separated by semi-colons and enclosed in parentheses.

FRAME is a stand-alone command, but also a qualifier for the graphical output commands PLOT, CONTOUR, FILL (alias for CONTOUR/FILL), SHADE, VECTOR and WIRE.

The saved animation frames are exactly the size and shape of the window from which they are created. Thus a large window results in a larger, slower animation that demands more disk space and memory to play back. The SET WINDOW/SIZE= command is generally used to specify minimally acceptable frame size.

See section "Advanced Movie-making" (p. 124), for more examples.

## Ch5 Sec3. DISPLAYING AN HDF MOVIE

Viewing a movie requires software which is not included with the Ferret distribution (although in some cases we have made the binary available in Ferret's anonymous ftp area). NCSA's X Data Slice reads HDF files and is available via anonymous ftp from NCSA. It requires about 1.7Mb of disk space. NCSA's ftp server is

ftp.ncsa.uiuc.edu  login id is "anonymous", give your e-mail address as the password

Consult the README files you will find there for instructions on obtaining X Data Slice. Other utilities from NCSA can also be used for animations.

## Ch5 Sec4. ADVANCED MOVIE-MAKING

## Ch5 Sec4.1. REPEAT command

The REPEAT command is quite flexible. It allows you to repeat a sequence of commands, not just a single command as in the basic example above. You can give the GO command as an argument to REPEAT. The following examples demonstrate these techniques.

**Note:** MODE VERIFY must be SET (this is the default state) for loop counting to work.

**Example 1**

Here we give multiple arguments to REPEAT; note the semi-colon separation and the parentheses. Note that FRAME, in this example, is used as a stand-alone command.

```
yes? REPEAT/L=1:12 (FILL SST; GO fland; FRAME/file=my_movie.mgm)
```

**Example 2**

In this example we use the REPEAT command to pan and zoom over a sea surface temperature field.

```
SET DATA coads_climatology
SET REGION/L=1
SET REGION/X=120E:60W/Y=45S:45N
SHADE sst; GO fland

! ZOOM
REPEAT/K=1:5 (SET REGION/DX=+8:-8/DY=+8:-8; SHADE sst; GO fland; FRAME)


! PAN
REPEAT/K=1:5 (SET REGION/DX=+5; SHADE/LEV=(20,30,.5) sst; FRAME)
```

**Example 3**

In this example the user calls setup_movie.jnl (text included below), title.jnl, which creates a title frame, then repeats main_movie.jnl (text included below) for each time step desired. Finally, the user adds a frame of credits at the end of the movie. Each of the scripts would end with the FRAME command (except setup_movie). Using GO scripts as arguments to REPEAT allows you to customize the plot with many commands before finally issuing FRAME, as the text of main_movie.jnl below demonstrates.

```
yes? ! make the movie
yes? GO setup_movie
yes? GO title
yes? REPEAT/L=1:12 GO main_movie
yes? GO credits

! Setup_movie.jnl
SET WINDOW/SIZE=.45/ASPECT=0.7
SET MOVIE/file=my_movie.mgm
SET DATA coads_climatology
SET REGION/X=130E:75W/Y=8S:8N
SET MODE CALENDAR:months
GO bold
PPL SHAKEY ,,.15,.2
PPL AXLEN 8.8,4.8

! Main_movie.jnl
FILL/SET_UP/LEVELS=(16,31,1) sst
PPL LABS; PPL TITLE
PPL FILL
LABEL 210,9.5,0,0,.22 @TRCOADS MONTHLY CLIMATOLOGY (1946-1989)
LABEL 210,-12,0,0,.22 @TRSEA SURFACE TEMPERATURE (DEG C)
LABEL 130,11,-1,0,.22 @TR'LAB4'
FRAME
```

**Note:** If you use the FILL command, we suggest that you use SHADE while customizing and fine-tuning your movie, then use FILL for the final run. SHADE is much faster.

## Ch5 Sec4.1.1. Initializing the color table

If you create a movie with a title frame, or a first frame which otherwise uses different colors than the rest of the movie, you should be aware of an HDF peculiarity: all the colors that you plan to use in your movie must be in the first frame, or else color behavior will be unpredictable when you animate.

To "reserve" the colors you need, use overlapping full-window viewports. Make a representative plot in the title frame, then cover over it with either a black or white rectangle and finally write the title text. Here is a script which initializes the color table while creating a title frame.

```
! define 3 identical full-frame viewports
DEFINE VIEW full1;  DEFINE VIEW full2;  DEFINE VIEW full3

! draw frame one of the movie in full color
SET VIEW full1
```

```
SET DATA coads_climatology
SHADE/LEVELS=(16,31,1)/L=1 sst                    ! dummy frame

! white-out over the picture
SET VIEW full2
GO setup_text
SHADE/PALETTE=white/NOLAB/NOKEY/i=1:2/j=1:2  (i+j)*0

!put on title frame labels (using [0,1] coordinate space)
SET VIEW full3
GO setup_text
PPL PLOT
LABEL .5,.7,0,0,.3 @TRMy Title
PPL ALINE 1,.2,.55,.8,.55
PPL ALINE 1,.2,.53,.8,.53
LABEL .5,.4,0,0,.2 @CRBy me

!capture the title frame and clean up
FRAME
GO cleanup_text
```

## Ch5 Sec4.1.2. Making movies in batch mode

Ferret, like other Unix applications, can be run in "batch" mode by redirecting standard input and output. Thus

```
ferret -unmapped <movie_commands.jnl >&movie.log&
```

will make a movie running in background mode based on the commands in file movie_commands.jnl logging standard output and standard error in file movie.log.

Note, however, that when used in this mode to make a movie Ferret will still require access to an X windows display (as in "setenv DISPLAY node:0"). To eliminate this requirement we recommend the use of the X11R6 "virtual frame buffer" (Xvfb). This application permits the movie frames to be generated in the absence of any physical display device. Consult your system manager for the availability of X11R6 for your system.

# Ch5 Sec5. CREATING GIF IMAGES

GIF is a highly compressed format suitable for single images. (Ferret will not directly create GIF89 animations.) The procedure for creating a GIF image is nearly identical to the creation of a single frame of an HDF file. The modification is generally just to select a file name with the ".gif" extension; Ferret will automatically sense this as a request to create a GIF-formatted image file. Alternatively, any file name can be used if the GIF format is specified explicitly using

```
FRAME/FORMAT=GIF
```

If a number of GIF images are created using the same file name Ferret will automatically rename subsequent versions with a version number. Thus a repeat loop can be used to generate many GIF images.

**Example:**

```
REPEAT/L=1:12(FILL sst; GO fland; FRAME/file=myimage.gif)
```

**Note:** In this mode of grabbing an image, Ferret creates a GIF file by requesting the image back from your screen (your X server). That means that the X server normally has to be configured as pseudo-color.

An alternative approach to creating GIF's (which does not share this restriction) is to invoke Ferret with the -gif command line switch "ferret -gif" (p. 6).

## Ch5 Sec6. CREATING MPEG ANIMATIONS

MPEG animations can be created from the outputs of the FRAME command—either HDF animation files or a sequence of GIF images. Various public domain utilities are available to perform the conversion from Ferret's output formats into MPEG animations. The routine hdf2mpeg (available in 1995 from ftp.ncsa.uiuc.edu in HDF/contrib/NCSA/HDF2MPEG) can be used to convert HDF files into MPEG animations; mpeg_encode (available from mm-ftp.CS.Berkeley.EDU in /pub/multimedia/mpeg/encode) can be used to convert sequences of GIF files. New and improved routines may have become available since the time of this writing. See further documentation on this topic in the FAQ file from the Ferret WWW home page.

# Chapter 6: CUSTOMIZING PLOTS

## Ch6 Sec1. OVERVIEW

Detailed control is possible over most aspects of Ferret graphical outputs. A custom modification will require the user to either add a qualifier to a Ferret command or communicate directly with the graphical package PPLUS, which is contained inside of Ferret. The most commonly used PPLUS commands are listed in the following sections of this chapter. Consult the PLOT PLUS for Ferret manual for complete command lists and the specifics of command syntax.

Ferret communicates with PPLUS by sending a sequence of commands to PPLUS (the command PPL ECHO ON causes the sequence of commands that Ferret sends to PPLUS to be logged in the file fort.41.). The user can give further commands to PPLUS directly using the Ferret command PPL (e.g., `yes? PPL AXLEN 10,7`). Some results can be attained in two ways—with either Ferret or PPLUS commands. However, the interaction of the two is complex and the inexperienced user may get unexpected results, so when possible, use only Ferret commands.[1]

PPLUS uses a deferred mode of output—various commands are given to PPLUS which describe the plot state but produce no immediate output; the entire plot is then rendered by a single command. Some plot states (e.g., axis labels) are set by Ferret with every plotted output; to customize these states it is necessary to use the /SET_UP qualifier (which sets up the plot inside of PPLUS) and then modify the state with direct PPL commands. Other plot states are never set by Ferret and, if modified at any time, remain in their specified state for all subsequent plots. Still other states are modified by Ferret only under special circumstances. Here is a very simple customization (Figure 6_1):

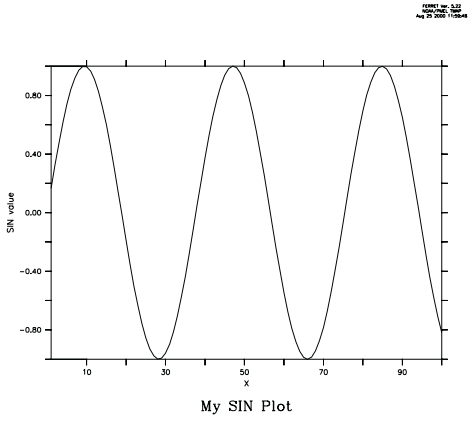



Figure 6_1

[1] Note that throughout this discussion a distinction has been made between Ferret commands and PPLUS commands. In reality, the user issues Ferret commands only. “PPLUS commands” in this context refers to PPLUS commands issued via the Ferret command PPL.

```
yes? PLOT/X=1:100/TITLE="My SIN Plot"/SET_UP sin(x/6)   !use /SET_UP
yes? PPL YLAB "SIN value"
yes? PPL PLOT
```

The examples throughout this chapter show how the /SET_UP qualifier on graphics commands can be used to delay rendering of a plot while the user modifies plot appearance with PPLUS commands.

Below is a list of PPLUS commands which are reset by Ferret:

| **PPLUS command** | **when reset by Ferret** |
|---|---|
| XFOR, YFOR | reset for every plot |
| XLAB, YLAB | reset for every plot |
| XAXIS, YAXIS | reset for every plot |
| LABS | reset for every plot |
| ALINE | reset for every plot |
| TAXIS OFF | reset for every plot |
| TITLE | reset for every plot |
| TICS | reset for every plot (small tic size, only) |
| WINDOW ON | reset for every plot |
| PEN 1,n | reset for every plot |
| LIMITS | reset for every plot |
| ORIGIN | reset by SET WINDOW/ASPECT and SET VIEWPORT; Y origin may be shifted to accommodate many line style keys |
| AXLEN | modified by SET WINDOW/ASPECT and SET VIEWPORT |
| VIEWPORT | modified by WIRE/VIEW |
| LEV | modified by CONTOUR and SHADE unless /LEVELS_SAME given |
| VECSET | modified by VECTOR unless /LENGTH_SAME given |
| WINDOW | modified for "fresh" plots but not for overlay plots |

# Ch6 Sec2. GRAPHICAL OUTPUT

## Ch6 Sec2.1. Ferret graphical output controls

| **Ferret command** | **Function** |
|---|---|
| CONTOUR | produces a contour plot of a single field |
| FILL | alias for CONTOUR/FILL; produces color-filled contour plot |
| PLOT | produces a line or symbol plot of one or more arrays |
| SHADE | produces a shaded representation (rectangular cells) |
| VECTOR | produces a vector arrow plot |

| Ferret command | Function |
|---|---|
| WIRE | produces a 3D wire frame plot |
| SET WINDOW | manipulates graphics windows |
| SET VIEWPORT | places graphics output into a sub-window (pane) |

## Ch6 Sec2.2. PPLUS graphical output commands

Whenever a plot is customized using /SET_UP to delay display, the plot will ultimately be rendered using a PPLUS graphical output command (not the Ferret counterpart). A customized contour or filled-contour plot is rendered with PPL CONTOUR, a wire frame plot with PPL VIEW and so on.

| Command | Function |
|---|---|
| CONTOUR | makes a contour plot |
| PLOT | plots x-y pairs for all lines of data |
| PLOTUV | makes a stick plot of vector data |
| SHADE | makes a shaded representation |
| VIEW | makes a wire frame plot |
| VECTOR | makes a plot of a vector field |

The graphical output command PLOTUV can be used to make stick plots easily, as the following time series example shows.

```
yes? SET DATA coads; SET REGION/X=180/Y=0/L=400:500
yes? PLOT/SET uwnd, vwnd
yes? PPL PLOTUV
```

## Ch6 Sec3. AXES

By default, Ferret displays X- and Y-axes with tics and numeric labels at reasonable intervals and a label for each axis. Time axes are also automatically formatted and used as needed. These axis features can be modified or suppressed using the following Ferret direct controls and PPLUS commands.

## Ch6 Sec3.1. Ferret axis controls

The following qualifiers are used with graphical output commands PLOT, VECTOR, SHADE, and CONTOUR to specify axis limits, tic spacing, and possible axis reversal:

**Ferret qualifers**

/XLIMITS, /YLIMITS, /NOAXIS

The /XLIMITS and /YLIMITS qualifiers use the syntax /XLIMITS=lo:hi:delta. Tic marks are placed every “delta” units, starting at “lo” and ending at “hi”. Every other tic mark is labeled. “delta” may be negative, in which case the axis is reversed.

The /NOAXIS qualifier removes both X and Y axes from the plot. This is particularly useful for plots using curvilinear coordinates (map projections) where the final axis values represent transformed axis values rather than world coordinates.

The following arguments to SET MODE and CANCEL MODE determine axis style (e.g., SET MODE CALENDAR:days) :

**Ferret arguments**

CALENDAR
LATIT_LABEL
LONG_LABEL

See the Commands Reference section of this manual (p. 241) for more information.

## Ch6 Sec3.2. PPLUS axis commands

| Command | Function |
|---|---|
| XAXIS* | controls numeric labeling and tics on the X axis (redundant with /XLIMITS) |
| YAXIS* | controls numeric labeling and tics on the Y axis (redundant with /YLIMITS) |
| AXATIC | sets number of large tics automatically for X and Y |
| AXLABP | locates or omits axis labels at top/bottom or left/right of plot |
| AXLEN** | sets axis lengths |
| AXLINT | sets numeric label interval for axes every nth large tic |
| AXLSZE | sets axis label heights |
| AXNMTC | sets number of small tics between large tics on axes |
| AXNSIG | sets number of significant digits in numeric axis labels |
| AXSET | allows omission of plotting of any axis |
| AXTYPE | sets axis type (linear, log, inv. log) for x- and y-axis |
| TICS | sets axis tic size and placement inside or outside axes |
| XFOR* | sets format of x-axis numeric labels |
| YFOR* | sets format of y-axis numeric labels |
| XLAB* | sets label of x-axis |
| YLAB* | sets label of y-axis |
| TXLABP | establishes time axis label position (or absence) |

| Command | Function |
|---|---|
| TXTYPE* | sets the style of the time axis |
| TXLINT* | specifies which time axis tics will be labeled |
| TXLSZE | sets height of time axis labels |
| TXNMTC | sets number of small tics between large tics on time axis |

* issued by Ferret with every relevant plot
** issued by Ferret upon SET WINDOW/ASPECT or SET VIEWPORT

## Examples

1) Plot with no axis labels (character or numeric) and no tics (Figure 6_2). (Equivalent to

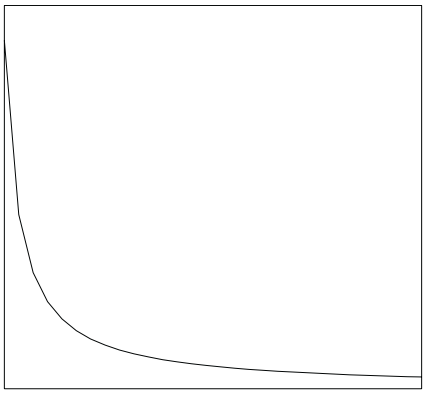


Figure 6_2

```
yes? GO box_plot PLOT/I=1:10/NOLABEL 1/i)

yes? PLOT/i=1:30/NOLABEL/SET 1/i
yes? PPL AXLABP 0,0                    !turn off numeric labels
yes? PPL TICS 0,0,0,0                  !suppress small and large tics
yes? PPL PLOT                          !render plot
yes? PPL TICS .125,.25,.125,.25        !reset tics to default
yes? PPL AXLABP -1,-1                  !reset numeric labels
```

2) customize x-axis label (Figure 6_3); XLAB always reset by Ferret)

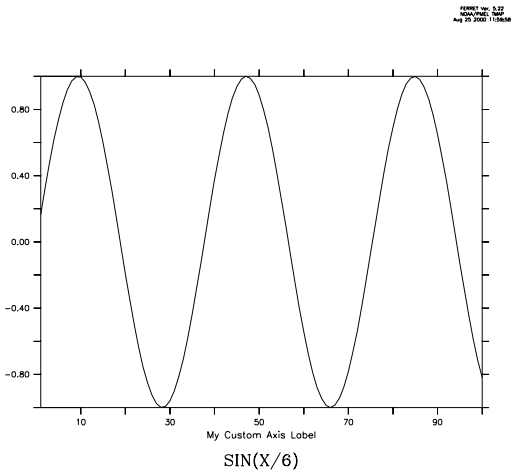



Figure 6_3

```
yes? PLOT/SET/i=1:100 sin(x/6)
yes? PPL XLAB My Custom Axis Label
yes? PPL PLOT
```

3) specify tic frequency for y axis

```
yes? PLOT/i=1:30/YLIM=0:1:.2 1/i
```

## Ch6 Sec3.3. Overlaying symbols on a time axis

To overlay symbols or mark-up on a plot which has a formatted time axis (dates and times) it is necessary to specify positions using the internal time encoding of that axis. Typically, the easiest way to achieve this is to define a variable, say TT, which is the time encoding. This example illustrates.

**Example:**

demonstrate PLOT/VS and POLYGON over time axes (Figure 6_4)

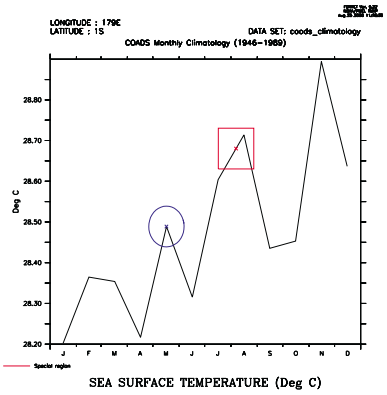



Figure 6_4

```
use coads_climatology
LET xsqr = {-1,1,1,-1}                          ! coordinates of a
unit square
LET ysqr = {-1,-1,1,1}

LET xcircle = COS(6.3*i[i=1:42]/40)    ! coordinates of unit circle
LET ycircle = SIN(6.3*i[i=1:42]/40)
! Notice the units of the time axis
show grid/l=1:3 sst

plot/x=180/y=0 sst     !  draw a  time series plot
let tt = T[gt=sst]          ! tt is the coordinates along the T axis
! place an "X" at the value exactly at 7-aug
! "@ITP" causes interpolation to exact location
let t0   = tt[t="7-aug-0000"@itp]
let val0 = sst[x=180,y=0,t="7-aug-0000"@itp]
plot/vs/over/nolab/sym=2/line=8 t0,val0

! put a box around the "X"
polygon/over/line=8/title="Special region" t0+500*xsqr, 0.05*ysqr+val0

! place an "X" on the data point nearest to 15-may
! Note that @ITP is absent, so behavior is set by MODE INTERPOLATE
let t1   = tt[t="15-may-0000"]
let val1 = sst[x=180,y=0,t="15-may-0000"]
plot/vs/over/nolab/sym=2/line=10 t1,val1
! put a circle around the "X"
plot/vs/over/line=10/nolab t1+500*xcircle,0.05*ycircle+val1
```

**Example (continued):**

mark-up over a Hofmuller diagram (Figure 6_5)

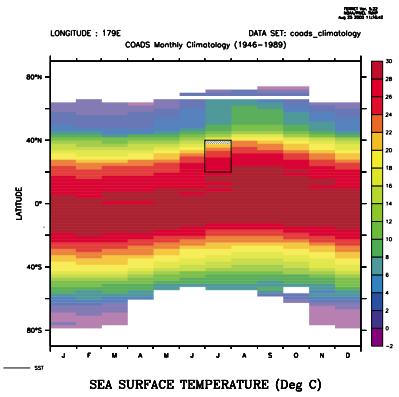



Figure 6_5

```
shade/x=180 sst                                  !  latitude vs time plot
let tlo = tt[t="1-jul-0000"@itp]
let thi = tt[t="1-aug-0000"@itp]
polygon/over/line=7/pal=gray/pat=lite_up_left_to_right {`tlo`, `thi`,`
thi`, `tlo`},  {20, 20, 40, 40}
```

## Ch6 Sec4. LABELS

Ferret, by default, produces labeled axes, a plot title, documentation about the plot axes normal to the plot, and a signature (current date and Ferret version number) when a plot is rendered. The /NOLABELS qualifier suppresses the plot title, the documentation and signature, but not the axis labels of independent axes; PPLUS commands XLAB, YLAB, and AXLABP control axis labels.

## Ch6 Sec4.1. Listing labels

The PPLUS command PPL LIST LABELS can be used to list the currently defined labels. For example,

```
yes? PPL LIST LABELS
@ACSEA SURFACE TEMPERATURE (Deg C)
@ASLONGITUDE
@ASLATITUDE

          XPOS       YPOS     HGT    ROT    UNITS
LAB 1   8.000E+00  7.200E+00 0.060     0   SYSTEM  @ASFERRET Ver. 4.40
LINE PT:    0.000E+00 0.000E+00  NO LINE     CENTER JUSTIFY LABEL
LAB 2   8.000E+00  7.100E+00 0.060     0   SYSTEM  @ASNOAA/PMEL TMAP
LINE PT:    0.000E+00 0.000E+00  NO LINE     CENTER JUSTIFY LABEL
```

```
 LAB 3  8.000E+00  7.000E+00 0.060    0  SYSTEM  @ASOct 22 1996 09:24
 LINE PT:    0.000E+00 0.000E+00  NO LINE     CENTER JUSTIFY LABEL
 LAB 4  0.000E+00  6.600E+00 0.120    0  SYSTEM  @ASTIME : 16-JAN
 LINE PT:    0.000E+00 0.000E+00  NO LINE     LEFT   JUSTIFY LABEL
.
.
.
```

The first three lines of output show the plot title, the X axis label, and the Y axis label. These labels are controlled by the PPL TITLE, PPL XLAB, and PPL YLAB commands, respectively. The three characters "@AS" indicate the font of the label—in this case "ASCII Simplex" (see the section in this chapter, "Fonts," p. 149).

Next is a table of "movable labels"—labels that were defined using the PPL LABS command. Labels are generally simpler to control with the GO unlabel and LABEL commands described in the following sections, rather than with the PPL LABS command.

Each label is described with two lines. The column headers refer to the first of the two. The coordinates of each label, (XPOS,YPOS), may be in units of "inches" or may be in the units of the axes. This is reflected in the UNITS field of the output, which will contain "SYSTEM" if the coordinates are in inches or "USER" if the coordinates are axis units. (The /NOUSER qualifier on the PPL LABS command is used to indicate that coordinates are being given in inches.) Coordinates are calculated relative to the axis origins. The PPL HLABS and PPL RLABS commands control label height and rotations, respectively.

The second line of the label description contains information about an optional line on the plot which can be used to point to the label (refer to the PPLUS command LLABS or see the section in this chapter, "Positioning labels using the mouse pointer," p. 141). At the end of this line is the text of the movable label.

## Ch6 Sec4.2. Adding labels

The Ferret command LABEL adds a label to a plot and takes the following arguments:

```
yes? LABEL xpos,ypos,center,angle,size text
```

where xpos and ypos are in user (axis) units, size is in inches, angle is in degrees (0 at 3 o'clock) and center is -1, 0, or +1 for left, center, or right justification. The label position will adjust itself automatically when the plot aspect ratio or the viewport is changed.

If you prefer to locate labels using inches rather than using data units issue the command

```
yes? LABEL/NOUSER xpos,ypos,...
```

Note, however, that the layout of a plot in inches—lengths of axes, label positions, etc.—shifts with changes in window aspect ratio (SET WINDOW/ASPECT) and with the use of

viewports. Labels specified using LABEL/NOUSER will need to be adjusted if the aspect ratio or viewport is changed.

**Notes:**

1) If you use the command PPL LABS instead of LABEL, be aware that when defining a new movable label, all lower-numbered labels must already be defined.

2) The Ferret command LABEL is an alias for PPL %LABEL. PPLUS does NOT consider a label created with LABEL a movable label. Consequently, no label number is assigned and the label cannot be manipulated as a movable label.

3) %LABEL is an unusual command in that the label appears on the plot immediately after the command is given, rather than being deferred. This has ramifications for the user who has multiple plot windows open and is in MODE METAFILE, since a metafile is not closed until a new plot is begun. If the user produces a plot in window B, and then returns to a previous window A and adds a label with LABEL, that label will appear on the screen correctly, but will be in the metafile corresponding to window B.

**Example**

```
yes? PLOT/I=1:100  sin(i/6)
yes? LABEL 50, 1.2, 0, 0, .2  @P2MY SIN PLOT
```

## Ch6 Sec4.3. Removing movable labels

Removing a movable label is a two step process: identifying the label number and then deleting the label. PPLUS internally refers to all movable labels with label reference numbers. The PPLUS command LIST LABELS will list the PPLUS labels and the text strings they contain. Then the user can use “GO unlabel n”, where n is the reference number, to delete a label.

**Example**

In this example we plot the same figure in two viewports, one plot with the default "signature," and one plot with the signature removed (Figure 6_6).

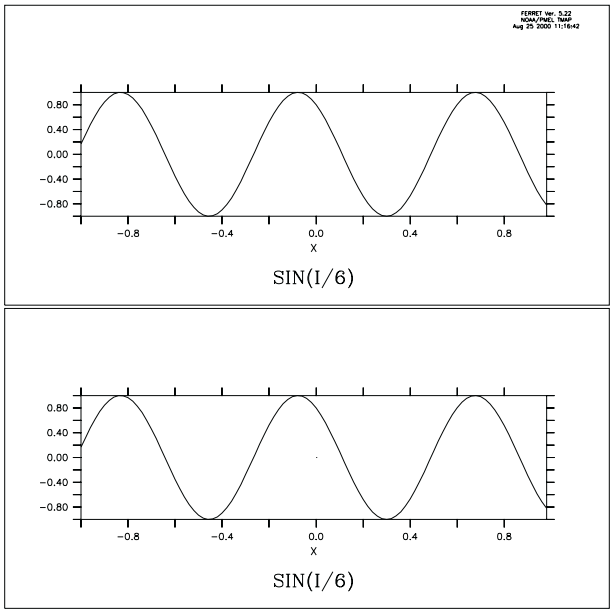



Figure 6_6

```
!upper viewport has a "signature"
yes? PPL BOX on
yes? SET VIEW upper
yes? PLOT/I=1:100 sin(i/6)

!in the lower viewport
!the signature has been removed
yes? SET VIEW lower
yes? GO unlabel 1
yes? GO unlabel 2
yes? GO unlabel 3
yes? PPL PLOT
yes? CANCEL VIEWPORT
```

## Ch6 Sec4.4. Axis labels and title

Special commands and special logic govern the labels of axes and titles. Use the PLOT+ commands XLAB, YLAB, and TITLE in conjunction with the Ferret plotting qualifier /SET_UP to modify the labeling choices that Ferret makes.

For two-dimensional plots (CONTOUR, FILL) Ferret will label the plot axes with the titles and units from the appropriate axes of the grid. The command SHOW GRID can be used to see the labels that will be used. The title will be the title of the variable (see SHOW VARIABLE, p. 325, and SHOW DATA/VARIABLE, p. 319) modified by the units and comments about transformations in parentheses.

For one-dimensional plots (PLOT) other than PLOT/VS the independent axis will be labeled using the title and units from the appropriate axis of the grid. The dependent axis will be labeled with the units of the variable being plotted. The title will be labeled as for two-dimensional plots.

For output of the PLOT/VS command the axes will be labeled with the titles of the variables (see SHOW VARIABLE, p. 325, and SHOW DATA/VARIABLE, p. 319) each modified by its units and comments about transformations in parentheses.

## Ch6 Sec4.5. Ferret label controls

In addition to LABEL (discussed above), Ferret controls include the /NOLABELS qualifier, which suppresses default plot title, documentation and signature, and /TITLE qualifier to graphical output commands PLOT, SHADE, CONTOUR, VECTOR, and WIRE:

**Ferret qualifiers**

/NOLABELS
/TITLE=

and arguments to SET MODE and CANCEL MODE:

**Ferret arguments**

ASCII_FONT
CALENDAR
LATIT_LABEL
LONG_LABEL

## Ch6 Sec4.6. PPLUS label commands

Ferret stores the text strings of the following labels in PPLUS symbols. The symbol names are:

| symbol name | label |
|---|---|
| LABTIT | title label |
| LABX | X axis label |
| LABY | Y axis label |
| LABn | nth movable label |

As stated above, PPLUS commands regarding movable labels are largely superceded by the Ferret command LABEL and “GO unlabel n”.

| Command | Function |
|---|---|
| LIST LABELS | shows the currently defined labels |
| LABS* | makes, removes, or alters a movable label |
| HLABS | sets height of each movable label |
| RLABS | sets angle for each movable label |
| LABSET | sets character heights for labels |
| LLABS | sets start position for and draws a line to a movable label |
| TITLE* | sets and clears main plot label |
| XLAB* | sets label of X axis |
| YLAB* | sets label of Y axis |

* issued by Ferret with every relevant plot

**Example**

This example customizes a plot using PPLUS label controls.

```
yes? PLOT/Z=20/I=1:100/SET_UP   z * sin(i/6)
yes? PPL LABS 4,48,0,0 @p2'lab4'
yes? PPL HLABS 4,.25
yes? PPL LABS/NOUSER 5,0,6.3,-1  *** Magnified SIN function ***
yes? PPL LABSET ,,,.35
yes? PPL PLOT
```

## Ch6 Sec4.7. Positioning labels using the mouse pointer

Often it is awkward precisely to position plot labels. Using the mouse pointer can simplify this as mouse clicks can be used to place labels and other annotations on plots. The full syntax of the LABEL command is

LABEL xpos, ypos, justify, rotate, height "text"

**xpos,ypos** are the (x,y) position of the label
**justify** = -1, 0, 1 for left, center, right justification — default = left
**rotate** is given in degrees counter-clockwise — default = 0
height is in "inches"
**text** to be plotted. This argument may include font and color specifications

Note that the LABEL /NOUSER qualifier is not relevant for mouse input.

If either of the first two arguments (label position) are omitted it is a signal that mouse input is desired. For example

```
yes? GO ptest
yes? LABEL "this is a test"
```

will wait for mouse input, using the indicated point as the lower left corner of the text string. Equivalent to this is

```
yes? LABEL ,,-1,0,.12 "this is a test"
```

Note that left justification will always be used in this mode, regardless of the value specified.

For mouse control over justification and/or to draw a line or arrow associating a label with a feature on the plot, explicitly omit the justification argument. Ferret will put up a menu requesting a selection of "Arrow", "Line", "Right", "Center", "Left". If Arrow or Line is selected two mouse inputs are required — the first indicating the feature to be marked, the second indicating the lower left corner of the text area. If "Right", "Center" or "Left" is specified the text will be justified accordingly.

Note that the mouse-driven LABEL command defines the symbols XMOUSE and YMOUSE and writes comments regarding their definitions into the current journal file (if any) as described under the WHERE alias.

The command (alias) WHERE requests mouse input from the user, using the indicated click position to define the symbols XMOUSE and YMOUSE in units of the plotted data. Comments which include the digitized position are also written to the current journal file (if open). The WHERE command can be embedded into scripts to allow interactive positioning of color keys, boxes, lines, and other annotations.

## Ch6 Sec4.8. Labeling details with arrows and text

Using the technique described in section 4.7 it is also simple to create a label with a line or arrow indicating a detail of a plot. Follow the procedure outlined above but select "Line" or "Fancy line" (arrow) from the menu that appears in the plot window. Then click on the detail which is to be labeled. The menu will appear again—this time select the justification and click on the label position.

To see the precise numerical coordinates of the arrow and label use the PPL ECHO ON command prior to the PPLUS command which redraws the plot. The endpoint coordinates of the arrow will appear as a comment line which begins with "C LLABS" in the echo file, fort.41. The coordinates of the label will appear as a comment line which begins with "C LABS". (Easily viewed with "spawn tail -2 fort.41".)

## Ch6 Sec5. COLOR

Ferret and PPLUS use colors stored by index. Storage indices 0 and 1 are used as window background and foreground colors. Indices 1–6 are reserved for lines. As the user makes SHADE and FILL requests, each color is assigned to the next available storage index beginning at 7, and that assignment is automatically "protected" when viewports or color overlays are added.

If your SHADE and FILL commands request more colors than there are storage indices (256), you will be informed with an error message and the color behavior may become unpredictable. For example, if you have multiple viewports defined within a window you may run out of color storage indices. If you are using the same color palette(s) in each viewport, you can free up indices by canceling the color protections with PPL SHASET RESET. See the examples later in this section for details on removing color protection. Currently, there is no way to ask PPLUS how many colors it is using in a plot.

The following discussion is divided into a treatment of text and line colors, and a discussion of shade and fill color.

## Ch6 Sec5.1. Text and line colors[2]

Line and text colors are regulated by use of storage indices 1–6, each index associated with a default color. It is possible to change the six available line colors with the PPLUS enhancements command COLOR. (See Plotplus Plus: Enhancements to Plotplus.) When you create a plot with multiple data lines, Ferret automatically draws each line in a different color. By default, axes, labels, and the first data line are all drawn in the same color. You can modify this behavior with the following Ferret and PPLUS commands.

See also the newer command qualifiers PLOT/COLOR= and PLOT/THICKNESS= in the Command Reference section (p. 280)

### Ch6 Sec5.1.1. Ferret color controls for lines

Plotted line colors can be set using

```
yes? PLOT/LINE=n
yes? VECTOR/PEN=n
yes? CONTOUR/PEN=n
```

[2] In the following discussion, "line color/thickness" is used as equivalent to "line style" for the sake of simplicity. However, if you are using a black and white printer, then the metafile translator will substitute a dash pattern for each line color. See Plotplus Plus: Enhancements to Plotplus to see monochrome line styles.

where “n” is an integer between 1 and 18.

## Ch6 Sec5.1.2. PPLUS text and line color commands

The PPLUS command PEN assigns a color and thickness index to a specified pen. (Note that as of Version 5.2 color and thickness can be assigned directly with qualifiers to the PLOT (p. 279), VECTOR, CONTOUR, or POLYGON commands with, for example, PLOT/COLOR=red/THICK=2 See the discussion of /COLOR and /THICKNESS in the Command Reference section, p. 279). The PPL command takes the form:

```
yes? PPL PEN pen_#, color_thickness
```

where pen_# is the PPLUS pen number and color_thickness is a color and thickness index. PPLUS uses different pens for different tasks. By default, color_thickness index 1 is assigned to pen 0. The following chart may be helpful.

| pen number | default color_thickness index | drawing task |
|---|---|---|
| 0 | 1 (black or white) | axes and labels |
| 1 | 1 (black or white) | first data line |
| 2 | 2 (red) | second data line |
| 3 | 3 (green) | third data line |
| 4 | 4 (blue) | fourth data line |
| 5 | 5 (cyan) | fifth data line |
| 6 | 6 (magenta) | sixth data line |

**Note:** Whether you plot several data lines simultaneously, or use the /OVERLAY qualifier on your Ferret commands, the color/thickness result will be the same. But the Ferret/PPLUS interaction is different. When Ferret plots multiple data lines simultaneously, PPLUS automatically cycles through pen numbers 1 through 6 combined with symbols. Type `GO line_samples` in Ferret to see the 36 different line styles. However, if you are using /OVERLAY for additional data lines, Ferret controls the color_thickness assigned to pen 1 and PPLUS draws each overlay line with pen 1.

Pen numbers range from 0 to 6, and color_thickness indices range from 0 to 18. The values 1 to 18 follow the formula:

color_thickness = 6 * (thickness - 1) + color

where thickness ranges from 1 to 3 and color from 1 to 6. Type “GO line_thickness” in Ferret to see actual colors and thicknesses.

The special color_thickness index 0 refers to the background color, which produces “invisible” lines that can be used as “white-out” for special purposes.

The following PPLUS commands use the color_thickness index.

| Command | Function |
|---|---|
| @Cnnn | uses color_thickness index "nnn" when embedded in a label |
| PEN | sets color_thickness index for each data line (see chart above) |
| LEV | sets color_thickness index for contour plot lines |

**Examples**

1) Ferret's default behavior—these two plots will look identical

```
yes? PLOT/i=1:10  1/i, 1/(i+3), 1/i + 1/(10-i)      !3 curves with 3
pens
yes? PLOT/i=1:10 1/i                                !first curve with
pen 1
yes? PLOT/OVER/i=1:10 1/(i+3)          !overlay with pen 1 (next index)
yes? PLOT/OVER/i=1:10 1/i+1/(10-i)     !overlay with pen 1 (next index)
```

2) select different colors for pens 0 and 1

```
yes? PLOT/i=1:10/SET 1/i
yes? PPL PEN 1 4          !assign color_thickness 4 to pen 1 (plot
curve)
yes? PPL PEN 0 3          !assign color_thickness 3 to pen 0 (axes &
labels)
yes? PPL PLOT             !render the plot
yes? PPL PEN 0 1          !reset pen 0 to default color_thickness (not\
                           reset by Ferret as is pen 1)
```

3) better way to do above plot:

```
yes? PLOT/i=1:10/LINE=4/SET 1/i  !include line style with qualifer
/LINE
yes? PPL PEN 0 3 ; PPL PLOT
yes? PPL PEN 0 1
```

## Ch6 Sec5.2. Shade and fill colors

Colors specified with the PPLUS SHASET command or in pallette files (also called spectrum files) contain pre-defined color palettes. With Ferret 5.0 there are now three ways to specify how colors are set in SHADE, FILL, and POLYGON plots: the earlier Percent RGB mapping, and also By_value and By_level.

The Percent method defines points along an abstract path in RGB color space that runs from 0 to 100 percent. The pallette file bluescale.spk, for example, contains these lines:

```
  0  0  0  95
100 95 95 95
```

The first number on a line is the percentage distance along the path in color space, and the following numbers are the percents of red, green, and blue, respectively. The actual colors used by SHADE or FILL are determined by dividing this abstract color scale into N equal increments, where N is the number of colors, and linearly interpolating between the red, green, and blue values from the neighboring SHASET percentage points.

For compatibility with older palette files, the Percent RGB mapping method is the default, and pre-5.0 palette files will be interpreted correctly. Palette files using Percent RGB mapping written out with Ferret 5.0 will have a slightly different format; for example the bluescale palette saved with Ferret 5.0 will look like this:

RGB_Mapping Percent

| SetPt | Red | Green | Blue |
|---|---|---|---|
| 0 | 0 | 0 | 95 |
| 100 | 95 | 95 | 95 |

The first line informs Ferret that the RGB mapping method is Percent. Lines beginning with an exclamation point are comments and ignored when read in—palette files created or modified using a text editor can contain comment lines as documentation.

The new RGB mapping method By_value uses color interpolation similar to the Percent method, with the significant difference that colors are based on the values of the variable being plotted rather than an abstract zero to 100 percent axis. When you use the same By_value palette in several plots, identical values of one variable will be represented by the same color in each plot. For example with the following palette, ocean_temp.spk:

RGB_Mapping By_value

| SetPt | Red | Green | Blue |
|---|---|---|---|
| –2.0 | 80.0 | 0.0 | 100.0 |
| 0.0 | 30.0 | 20.0 | 100.0 |
| 10.0 | 0.0 | 60.0 | 30.0 |
| 20.0 | 100.0 | 100.0 | 0.0 |
| 30.0 | 100.0 | 0.0 | 0.0 |
| 35.0 | 60.0 | 0.0 | 0.0 |

a particular temperature, say 25 degrees, will have the same color on a SHADE or FILL plot with levels ranging from 0 to 30, and on a plot with levels between 20 and 30 degrees.

The second new RGB mapping method By_level allows the user to select the precise color to be used at each level in SHADE and FILL plots. Unlike the other methods, no interpolation of RGB values is done. Colors specified in the palette will be used exactly as defined. If there are

more SHADE or FILL levels than colors specified, the color palette will repeat. In the following palette, by_level_rainbow.spk,

RGB_Mapping By_level

| Level | Red | Green | Blue |
|---|---|---|---|
| 1 | 80.0 | 0.0 | 100.0 |
| 2 | 30.0 | 20.0 | 100.0 |
| 3 | 0.0 | 60.0 | 30.0 |
| 4 | 100.0 | 100.0 | 0.0 |
| 5 | 100.0 | 0.0 | 0.0 |
| 6 | 60.0 | 0.0 | 0.0 |

for example, with 6 colors defined and used in a plot with 10 levels, the colors used at each plot level will be as follows:

| Plot level | Color |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 1 |
| 8 | 2 |
| 9 | 9 |
| 10 | 10 |

## Ch6 Sec5.2.1. Ferret shade and fill color controls

By default, Ferret will use the PPLUS spectrum file default.spk for shades and fills (normally default.spk is a Unix soft link to rnb.spk). Ferret comes with many color palettes. The UNIX command "Fenv" lists the environment variable $FER_PALETTE which is a list of paths to be searched for palette files (the palette file names all end in .spk). The UNIX command "Fpalette" allows you to find and examine these files (type "Fpalette -help" at the Unix prompt). You can easily create your own palette files with a text editor.

Use the Ferret qualifier /PALETTE= with Ferret graphical output commands CONTOUR/FILL and SHADE to specify a color palette. See the section in this chapter, "Contouring," p. 154, for details on the CONTOUR qualifier /LEV, which controls colors and dash patterns, as well as sets contour levels.

**Ferret qualifiers**

/PALETTE= (alias for PPL SHASET SPECTRUM=)
/LEV=

PALETTE is also a stand-alone command alias; it sets a new default color palette.

Be aware that when you use /PALETTE= in conjunction with /SET_UP, the color spectrum you specify becomes the new default palette; to restore the default palette use command PALETTE with no argument.

## Ch6 Sec5.2.2. PPLUS shade color commands

| Command | Function |
|---|---|
| SHASET | sets colors used by SHADE |

SHASET is an enhancement of PPLUS designed for Ferret. You can specify a color spectrum, save a spectrum, change an individual color in the spectrum, or remove the protection (PPL SHASET RESET) for colors already on the screen. See Plotplus Plus: Enhancements to Plotplus for more information.

If you need precise control over each individual RGB color on your plot, run "GO exact_colors", which contains instructions on modifying individual colors in a palette using SHASET.

**Examples**

1) look at the relief of the earth's surface

```
yes? SET DATA etopo120
yes? SHADE rose                 !Ferret's default behavior
yes? SHADE/PAL=land_sea rose    !emphasize land and sea with palette
```

2) Perhaps you would like to compare two topography resolutions. To illustrate what happens when you use more colors than are available, request an excessively large number of levels:

```
yes? SET DATA etopo120
yes? SET REGION/Y=-20:20
yes? SET VIEWPORT UPPER                     !upper half
yes? SHADE/LEV=(-8000,8000,100) rose        !160 colors, default palette
yes? SET VIEWPORT LOWER                     !lower half
yes? SET DATA etopo20                       !high resolution
yes? SHADE/LEV rose[d=etopo20]              !another 160 colors (320 >
256!)
yes? CANCEL VIEWPORT

PPL+ error: You're attempting to use more colors than are available.
            Using SHASET RESET to re-use protected colors may help.
```

If you reuse the same palette, as in this example, you can issue PPL SHASET RESET after the first plot and plot the second picture without error:

```
yes? SET DATA etopo120
yes? SET REGION/Y=-20:20
yes? SET VIEWPORT UPPER
yes? SHADE/LEV=(-8000,8000,100) rose
yes? SET VIEWPORT LOWER
yes? PPL SHASET RESET                    !reuse color storage indices
yes? SET DATA etopo20
yes? SHADE/LEV rose[d=etopo20]
yes? CANCEL VIEWPORT
```

## Ch6 Sec6. FONTS

## Ch6 Sec6.1. Ferret font controls

By default, Ferret produces all plot labels using the fonts ASCII Simplex (code AS) and ASCII Complex (code AC). For upper and lower case letters these fonts are identical to the fonts Simplex Roman (SR) and Complex Roman (CR), respectively. In addition, however, fonts AS and AC include the complete set of ASCII punctuation characters and ignore the special PPLUS interpretations of the characters "^" (superscript), "_" (subscript), and "@" (change font or pen). Using a text editor, the ESCAPE character (decimal 27) may be inserted before the special characters to restore their special interpretation.

The Ferret command CANCEL MODE ASCII causes Ferret to generate PPLUS labels which have the font unspecified. When the font is unspecified the PPLUS command DFLTFNT determines the default font and PPLUS responds to the special characters "^", "_", and "@". SET MODE ASCII restores normal font behavior.

## Ch6 Sec6.2. PPLUS font commands

| Command | Function |
|---|---|
| DFLTFNT | Sets default character font for all labeling. |
| @AB | In a label string, selects the font for which AB is a two-letter abbreviation (i.e., @CI for complex italic—see PPLUS manual for fonts). |

Note that many ASCII punctuation characters are printable only in ASCII simplex and complex fonts. In all other fonts these characters "@", "^", and "_" have special meanings: @ = font change; ^ = superscript; _ = subscript.

**Examples**

1) axis labels in custom fonts (Figure 6_7)

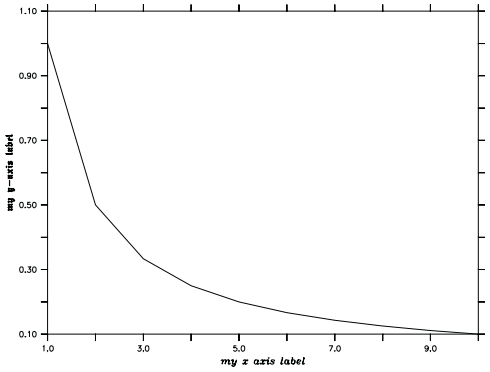



Figure 6_7

```
yes? PLOT/SET/i=1:10/NOLAB 1/i
yes? PPL XLAB @CImy x-axis label
yes? PPL YLAB @GEmy y-axis label
yes? PPL PLOT
```

2) set default font for all labeling (Figure 6_8)

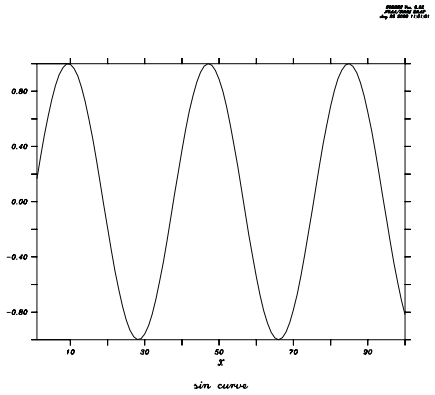



Figure 6_8

```
yes? CANCEL MODE ASCII
yes? PPL DFLTFNT CS      !complex script
yes? PLOT/I=1:100/TITLE="sin curve"  sin(i/6)
yes? SET MODE ASCII
yes? PPL DFLTFNT SR      !numeric axis labels unaffected by SET MODE
ASCII
```

# Ch6 Sec7. PLOT LAYOUT

## Ch6 Sec7.1. Ferret layout controls

Layout of plots can be controlled with commands which modify window size and aspect ratio, and viewports.

**Ferret command**

SET WINDOW/SIZE=/NEW/ASPECT=
DEFINE VIEWPORT/XLIMITS=/YLIMITS=/TEXT= view_name
SET VIEWPORT view_name
CANCEL VIEWPORT

### Ch6 Sec7.1.1. Viewports

A viewport is a sub-rectangle of a full window. Viewports can be used to put multiple plots onto a single window. Issuing the command SET VIEWPORT is best thought of as entering "viewport mode." While in viewport mode all previously drawn viewports remain on the screen until explicitly cleared with either SET WINDOW/CLEAR or CANCEL VIEWPORT. If multiple plots are drawn in a single viewport without the use of /OVERLAY the current plot will erase and replace the previous one; the graphics in other viewports will be affected only if the viewports overlap. If viewports overlap the most recently drawn graphics will always lie on top, possibly obscuring what is underneath. By default, the state of "viewport mode" is canceled. A number of the most commonly desired viewports are pre-defined.

### Ch6 Sec7.1.2. Pre-defined viewports

| Name | Description |
|---|---|
| FULL | full window |
| LL | lower left quadrant of window |
| LR | lower right quadrant of window |
| UR | upper right quadrant of window |
| UL | upper left quadrant of window |
| RIGHT | right half of window |
| LEFT | left half of window |
| UPPER | upper half of window |
| LOWER | lower half of window |

### Example: Graphics Viewports

Plot four variables from coads_climatology into the four quadrants of a single window (Figure 6_9).

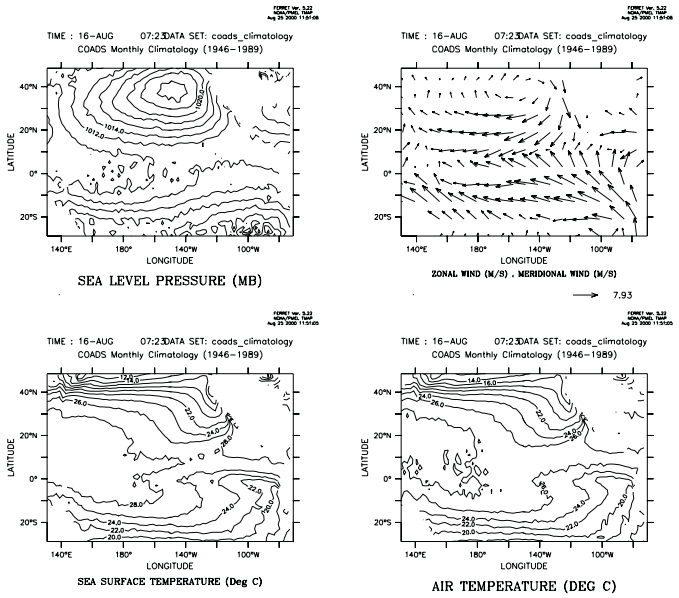



Figure 6_9

```
yes? SET DATA coads_climatology
yes? SET REGION/@W/L=8
yes? SET VIEWPORT LL
yes? CONTOUR sst                          !sea surface temperature
yes? SET VIEWPORT LR
yes? CONTOUR airt                         !air temperature
yes? SET VIEWPORT UL
yes? CONTOUR slp                          !sea level pressure
yes? SET VIEWPORT UR
yes? VECTOR/XSKIP=4/YSKIP=4 uwnd,vwnd     !zonal wind, meridional wind
yes? CANCEL VIEWPORT
```

## Ch6 Sec7.1.3. Advanced usage of viewports

For the purposes of defining viewports, a graphics window is considered to have length 1 and height 1. All viewport commands refer to positions relative to the current aspect ratio of the window. Thus,

```
yes? DEFINE VIEWPORT/XLIM=.5,1/YLIM=.5,1  V5
```

will locate the origin of viewport V5 in the upper right of the output window regardless of the shape of the window.

```
yes? DEFINE VIEWPORT/XLIM=0.,1/YLIM= 0,.3 V1
yes? DEFINE VIEWPORT/XLIM=0.,1/YLIM=.3,.6 V2
yes? DEFINE VIEWPORT/XLIM=0.,1/YLIM=.6,.9 V3
```

defines three viewports; each takes a third of the height of the page, and the entire width.

The qualifiers /XLIMITS=x1,x2 and /YLIMITS=y1,y2 allow the user to specify a portion of the graphics window to be the defined viewport. The arguments must be values between [0,1] (NOT world coordinates). x1 and x2 indicate the lower and upper bounds for the length of the window to be defined as the viewport; y1 and y2 serve an analogous purpose for height.

The /TEXT=n qualifier allows the user control over the shrinkage or enlargement of text on the plot. A value of /TEXT=1 indicates that the text size should be the same as it is on the full screen output. If a value less than 1 is specified the text will shrink. If a value is not specified Ferret chooses a value appropriate to the viewport size. Acceptable values are $0 < n < \text{inf.}$ but only values up to about 2 yield useful results.

## Ch6 Sec7.2. PPLUS layout commands

| Command | Function |
|---|---|
| ORIGIN | sets distance of plot origin from lower left corner |
| BOX | controls drawing of a box around the plotting area |
| CROSS | controls drawing of lines through (0, 0) on graph |
| ROTATE | rotates plot by 90 degrees on screen and plotter |
| AXLEN | sets axis lengths |
| SHAKEY | locates the color key |
| VECKEY | locates the vector key |
| AXSET | includes/excludes particular axes |
| SIZE | sets the overall size of the graphics window |

## Ch6 Sec7.3. Controlling the white space around plots

The location and size of the axis rectangle within the viewport or window determines the amount of white space surrounding a plot. Complete control over this is possible using low level controls, DEFINE VIEWPORT/TEXT_PROMINENCE, PPL ORIGIN, and PPL AXLEN, but these commands are sometimes awkward to work with. A simpler strategy is to use the GO tool

```
yes? GO margins
```

When given without arguments this command will report the amount of white space surrounding a plot. With arguments it will adjust the axis origins and lengths according to the requested margins. Try the Unix command

```
> Fgo -more margins
```

for further documentation.

## Ch6 Sec8. CONTOURING

## Ch6 Sec8.1. Ferret contour controls

The following qualifiers to the Ferret command CONTOUR allow customization of a contour plot.

| Qualfier | Function |
|---|---|
| /FILL | produces a color-filled contour plot (command FILL is an alias for CONTOUR/FILL) |
| /LEVELS | specifies contour levels, dash patterns, line thickness and color |
| /KEY | turns on display of color key for color-filled contour plots (default) |
| /NOKEY | turns off display of color key for color-filled plots |
| /NOAXIS | turns off display of X and Y axes (useful for map projections) |
| /LINE | adds contour lines to a color-filled plot (lines replace key) |
| /PALETTE= | specifies a color palette for color-filled contour plot |
| /PEN= | sets line style for contour lines (same arguments as PLOT/LINE=. See the section in this chapter, “Text and Line Colors,” p. 143.) |

### Ch6 Sec8.1.1. /LEVELS qualifier

The /LEVELS qualifier is a powerful and multi-functional tool. It takes the form **/LEVELS=levels_descriptor**

**/LEVELS** without an argument /LEVELS instructs Ferret to reuse CONTOUR or SHADE levels from the last CONTOUR or SHADE plot

**/LEVELS=n** specifying a simple numerical argument such as /LEVELS=25 instructs Ferret to select approximately 25 levels automatically, based upon the limits of the data to be plotted

**/LEVELS=nC** (centered levels) appending a "C" to the suggested number of levels instructs Ferret to select levels which are centered about the zero level. Such levels are suitable for zero-symmetric quantities such as anomalies and velocity components.

**/LEVELS=x.xD** (delta levels) Use of "D" as a suffix instructs Ferret to use the preceding value as the delta value between contour levels. Thus /LEVELS=0.25D will cause Ferret to select contour levels that span the range of the data to be contoured with a delta value of 0.25 between contour levels. The "D" and "C" notations can be combined. For example, /LEVELS=0.25DC instructs Ferret to create zero-centered levels with a delta of 0.25 spanning the range of the data.

**/LEVELS=(lo, hi, delta)**
or
**/LEVELS=(lo, hi, delta, ndigits)**
or
**/LEVELS=(value)**

where ndigits is the number of decimal places to use on contour levels as
-1 for integer format
or
-3 to omit numerical labels

**Examples**

Note that by default the contour lines of negative values will be dashed and the zero contour will be a heavy (DARK) line. See also (p. ) for selecting color and thickness with the PEN option, below.

```
/LEVELS=(-20,10,2)        ! basic low,high,delta
/LEVELS=(5)               ! a single level at 5
/LEVELS=(-20,10,2,-3)     ! suppress numerical contour labels
/LEVELS=40                ! approximately 40 automatically selected levels
/LEVELS=40C               ! approximately 40 automatic levels centered
                          equally about zero
/LEVELS=0.2D              ! automatic levels with a delta value of 0.2
/LEVELS=0.2DC             ! automatic zero-centered levels with a delta
                          value of 0.2
```

Refinements to the basic levels may be applied using the syntaxes below. If blanks are included, surround the entire levels descriptor in double quotation marks.

To request additional levels, simply append additional (lo, hi, delta) and/or (value) specifiers.

```
/LEVELS="(-100) (-10,10,2) (100)"      ! focus on -10 to10 range,  but catch
                                      outliers
```

To specify the line type as dark (heavy line), append DARK(lo, hi, delta) or DARK(value). Similar syntax can be applied to LINE (solid, thin) or DASH.

```
/LEVELS="(-100,100,5)DARK(-100,100,25)"    ! heavy line on  multiples of 25
/LEVELS="(0,10,2) DASH(2,10,2)"            ! use dashed lines for positive
                                           values
```

To remove selected levels, append the specifier DEL(lo, hi, delta) or DEL(value).

```
/LEVELS="(-10,10,2) DEL(0)"          ! -10 to 10 by 2's with the zero con-
                                     tour removed
```

To specify the color_thickness index of contour lines (see the section in this chapter, “Color, ” p. 143, for a discussion of color_thickness indices), append PEN(lo, hi, delta, index).

```
/LEVELS=(0,1,.2) PEN(.6,1,.2,2)         ! use pen #2 (red) for the upper con-
                                        tour levels
/LEVELS="(-100,100,10)                  ! Use Pen 2 (red) for negative levels
PEN(-100,-10,10,2) PEN(10,100,10,4)"    and  pen 4 (blue) for positive levels.
```

## Ch6 Sec8.2. PPLUS contour commands

| Command | Function |
|---|---|
| CONPRE | sets prefix for contour labels (usually a font, e.g., “@TR”) |
| CONPST | sets suffix for contour labels (usually units, e.g., “cm”) |
| CONSET | controls various aspects of contour labels and curves (see below) |

CONSET is a modified version of the PPLUS command. Two new parameters have been added—“spline_tension” and “draftsman”. “spline_tension” controls a spline fitting routine for contour lines, and is primarily used in conjunction with the narc parameter. The new parameter “draftsman” enables the user to specify horizontally oriented contour labels (draftsman style) or the default, labels oriented along contour lines. Arguments for CONSET are as follows:

```
CONSET
hgt,nsig,narc,dashln,spacln,cay,nrng,dslab,spline_tension,draftsman
```

**hgt** = height of contour labels. default=.08 inches

**nsig** = no. of significant digits in contour labels. default=2

**narc**= number of line segments to use to connect contour points. default=1

**dashln**= dash length of dashes mode. default=.04 inches

**spacln**= space length of dashes mode. default=.04 inches

**cay** This argument has no effect on gridded data. It is documented in PLOT PLUS for Ferret User's Guide and also in the discussion of objective analysis under command USER in the Commands Reference section of this manual.

**nrng** This argument has no effect on gridded data. It is documented in PLOT PLUS for Ferret User's Guide and also (as parameter "rng") in the discussion of objective analysis under command USER in the Commands Reference section of this manual.

**dslab**= nominal distance between labels on a contour line. default=5.0 inches.

**spline_tension**= a real value that affects the fit of the contour line. default=0. This parameter is only applied if narc is greater than 1. Otherwise, straight lines are drawn between data points and no interpolated points are contoured. This value indicates the curviness desired.

abs(spline_tension) is nearly zero (e.g., .01). The resulting curve is approximately a cubic spline.
abs(spline_tension) is large (e.g., 10.). The resulting curve is nearly a polygonal line.
spline_tension = 0. The resulting curve is a cubic spline (the default algorithm in ppl).

A typical value for spline_tension is 1, and the typical useful range of values is .01 to 10.

**draftsman**= a real value that controls the label format. default = 0.
0. = original label style—labels oriented along contour arcs
> 0. = draftsman label style—labels oriented horizontally on the page
< 0. = reserved for future use

**Examples**

Run the demonstration on custom contouring for many examples of label styles, contour line styles (color, thickness dash pattern), and contour intervals— `yes? GO custom_contour`

1) Color-filled contour plot of sea surface temperature

```
yes? SET DATA coads_climatology
yes? SET REGION/@t/l=6          !specify tropical Pacific, month 6
yes? SET VIEWPORT upper
yes? FILL sst                   !filled contour plot
yes? SET VIEWPORT lower
yes? FILL/LINE sst              !make the plot with contour lines
```

2) Let's improve on the earlier example (5.2.2) of shaded bathymetry with blue palette

```
yes? SET DATA ETOPO60
yes? LET/TITLE="Surface relief x1000 (meters)" r1000 rose/1000
yes? FILL/PAL=ocean_blue/LINE/LEV=(-8,-1,1,-3)LINE(-8,-1,1,-3)/PEN=4
r1000
```

Here is a breakdown of the final command line:

| | |
|---|---|
| FILL | color-filled contour plot (alias for CONTOUR/FILL) |
| PAL | specifies color palette for fill colors |
| LINE | specifies that contour lines be overlaid on the filled plot (in lieu of a key) |
| LEV | first arg specifies contour levels without numerical labels, next requests solid lines (dashed lines are the default for negative contour values) |
| PEN | assigns line style 4 (blue) to contour lines |

## Ch6 Sec9. PPLUS SPECIAL SYMBOLS

PPLUS defines a number of global symbols which are available to the user with SHOW SYMBOL They are documented in the PPLUS Users Guide, section 7.3, and listed here. These are not defined until associated plot commands have been issured. Also note that the user cannot redefine the value of these symbols.

**Example: draw a plot and examine some of the symbols**

```
yes? plot/i=1:10 1./i

yes? SHOW SYMBOL ppl$xlen
PPL$XLEN = "8.000"
    ! Try to show an undefined variable (no response)
yes? SHOW SYMBOL ppl$lf_var

yes? SHOW SYMBOL ppl$line_count
PPL$LINE_COUNT = " 1"
```

| SYMBOL | COMMAND | DESCRIPTION |
|---|---|---|
| PPL$EOF | RD,RWD,SKP | "YES" if an EOF (end of file) was read. |
| PPL$FORMAT | FORMAT | The current format. |

| | | |
|---|---|---|
| PPL$HEIGHT | SIZE | Height of the box. |
| PPL$INPUT_FILE | RD,SKP,RWD | The current input file. |
| PPL$LF_A | LINFIT | Constant from fit y= a + b*x |
| PPL$LF_A_STDEV | LINFIT | Standard error of A. |
| PPL$LF_B | LINFIT | Constant from fit. |
| PPL$LF_B_STDEV | LINFIT | Standard error of B. |
| PPL$LF_R2 | LINFIT | Regression coefficient squared. |
| PPL$LF_RES_VAR | LINFIT | Residual variance. |
| PPL$LF_VAR | LINFIT | Total variance. |
| PPL$LINE_COUNT | - | The number of the last line read. |
| PPL$PLTNME | PLTNME | The name of the plot file. |
| PPL$RANGE_INC | %RANGE | See Advanced Commands Chapter |
| PPL$RANGE_HIGH | %RANGE | See Advanced Commands Chapter |
| PPL$RANGE_LOW | %RANGE | See Advanced Commands Chapter |
| PPL$TEKNME | TEKNME | The name of the tektronix file. |
| PPL$VIEW_X | VPOINT | X viewpoint |
| PPL$VIEW_Y | VPOINT | Y viewpoint |
| PPL$VIEW_Z | VPOINT | Z viewpoint |
| PPL$WIDTH | SIZE | Width of the box. |
| PPL$XFACT(n) | TRANSXY | Xfact for line n. |
| PPL$XLEN | AXLEN | Length of X axis. |
| PPL$XOFF(n) | TRANSXY | Xoff for line n. |
| PPL$XORG | ORIGIN | Distance between origin and left edge. |
| PPL$XFIRST(n) | - | X value for first data point in line n. |
| PPL$XLAST(n) | - | X value for last data point in line n. |
| PPL$XMAX | RD | Xmax of contour grid |
| PPL$XMIN | RD | Xmin of contour grid |
| PPL$XMAX(n) | - | Xmax for valid data in line n. |
| PPL$XMIN(n) | - | Xmin for valid data in line n. |
| PPL$YFACT(n) | TRANSXY | Yfact for line n. |
| PPL$YLEN | AXLEN | Length of Y axis. |
| PPL$YOFF(n) | TRANSXY | Yoff for line n. |
| PPL$YORG | ORIGIN | Distance between origin and bottom edge. |
| PPL$YFIRST(n) | - | Y value for first data point in ine n. |
| PPL$YLAST(n) | - | Y value for last data point in line n. |

| | | |
|---|---|---|
| PPL$YMAX | RD | Ymax of contour grid |
| PPL$YMIN | RD | Ymin of contour grid |
| PPL$YMAX(n) | - | Ymax for valid data in line n. |
| PPL$YMIN(n) | - | Ymin for valid data in line n. |
| PPL$ZMAX | - | Zmax for valid contour data. |
| PPL$ZMIN | - | Zmin for valid contour data. |

# Ch6 Sec10. MAP PROJECTIONS AND CURVILINEAR COORDINATES

## Ch6 Sec10.1. Three-argument (curvilinear) version of SHADE, FILL, CONTOUR, and VECTOR

The SHADE, FILL, CONTOUR and VECTOR commands now have a 3-argument mode which allows them to create output in "curvilinear" coordinates. This allows for easy generation of output plots using sigma coordinates as well as the application of various map projections. A typical command line entry will look like:

```
yes? SHADE sst, x_page, y_page
```

where the second and third arguments, `x_page(i,j)` and `y_page(i,j)`, must be (at least) 2-dimensional grids which specify the X page (horizontal) position and Y page (vertical) position for each `(i,j)` index pair. The page positions may be in any units; Ferret will scale the plot according to the ranges of values in the position fields.

**Note:** The default axis labeling for the 3-argument commands will be the ranges of the position fields: inappropriate when map projections are being used. The /NOAXIS qualifier is provided for this purpose.

The /NOAXIS qualifier causes the axes and axis labels to be omitted from the plot. (See the AXSET command in the PLOT+ Users Guide). The qualifier has been added to support the curvilinear coordinate and map projection capabilities of the 3-argument versions of SHADE, FILL, CONTOUR and VECTOR in which linear axes are inappropriate.

Note that if the /SET_UP qualifier is used in conjunction with /NOAXIS a Ferret state is altered such that future plots will be drawn without axes. Ferret will warn you of this and coach you to use PPL AXSET 1, 1, 1, 1 to restore normal axis drawing.

## Ch6 Sec10.2. Gridded data sets on curvilinear coordinates

If a given gridded variable is defined on a curvilinear coordinate system, then one need only provide the X and Y coordinate fields in the 3-argument SHADE or FILL command to accurately depict the field. For example, if a data set contained a variable TEMP, which was Nx x Ny in the longitude-latitude plane, and the data set also contained variables LON_POSITION and LAT_POSITION of the same size, then the command:

```
yes? SHADE TEMP, LON_POSITION, LAT_POSITION
```

would render the curvilinear plot.

## Ch6 Sec10.3. Layered (sigma) coordinates

The capability to render curvilinear coordinates allows Ferret to display sigma coordinate fields without interpolating or regridding the variable to be displayed.

In this example the variable flow is defined on the gg grid where the Z axis is in layers. To display the field we need only create multidimensional fields specifying the relative positions of (i,j) pairs and use the new curvilinear coordinate commands (Figure 6_10):

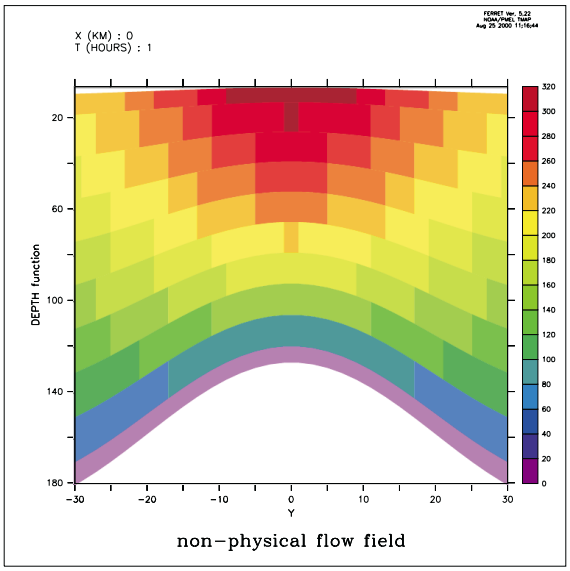



Figure 6_10

```
let depth = h[k=@rsum]-h/2
set variable/title="DEPTH function"/unit=meters depth
! regrid 'Y' to the data grid
let ygg = y[g=gg]
set variable/title="Y"/unit=kilometers ygg
shade flow[x=0,l=1], ygg, depth[x=0,i=1]
```

For a detailed example illustrating the use of curvilinear coordinates to analyze sigma-coordinate fields see the Ferret FAQ entry, How to handle sigma coordinate output in Ferret.

## Ch6 Sec10.4. Map Projections

Along with general capabilities for curvilinear coordinates, version 4.9 of Ferret and later provide a series of scripts for many common map projections.

Each map projection script will create the following variables:

| | |
|---|---|
| `mp_central_meridian` | central longitude calculated from the currently set region |
| `mp_standard_parallel` | central latitude calculated from the currently set region |
| `x_page` | two dimensional array mapping X world coordinates to page coordinates |
| `y_page` | two dimensional array mapping Y world coordinates to page coordinates |
| `mp_mask` | mask two hide “back side” data in orthographic or other 3-D projections |

### Ch6 Sec10.4.1. Using Map Projection scripts

To create output with a particular map projection you must do the following:

1. `set grid` for the variable you wish to plot
2. run the map projection script
3. adjust the window aspect ratio (if desired)
4. multiply the variable of interest by mp_mask (required for “3-D” projections)
5. give the three-argument plotting command

**Example:**(Figure 6_11**)**

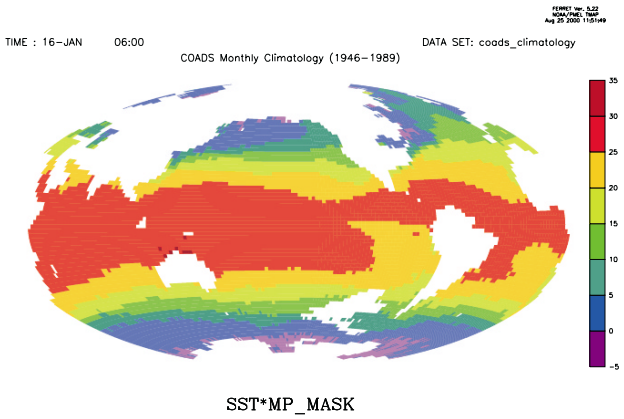



Figure 6_11

```
yes? use coads_climatology
yes? set region/l=1
yes? set grid sst
yes? go mp_hammer
yes? go mp_aspect
yes? shade/noaxis sst*mp_mask, x_page, y_page
```

## Ch6 Sec10.4.2. Overlays with Map Projections

Overlays can be drawn once a map projection script has been run. To add a filled land mask, sea level pressure and wind vectors onto our SST map we would issue the following commands (Figure 6_12):

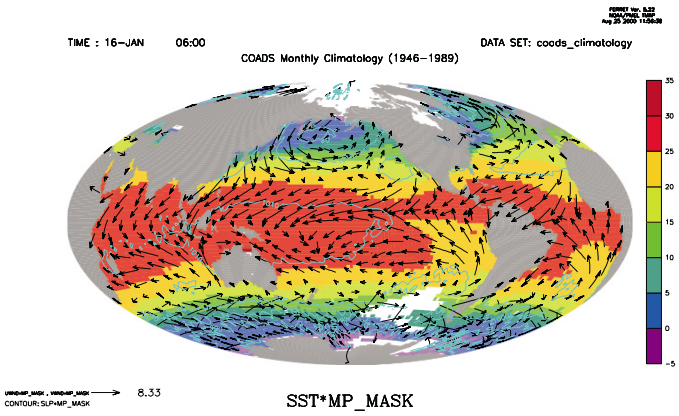



Figure 6_12

```
yes? set grid uwnd
yes? go mp_fland
yes? vector/over/pen=1 uwnd*mp_mask, vwnd*mp_mask, x_page, y_page
```

```
yes? set grid slp
yes? contour/over/pen=5 slp*mp_mask, x_page, y_page
```

Note that the 4-argument (curvilinear) VECTOR command is unsupported (as of Version 5.2) and is known not to work for irregular grids. The command is provided to help give a qualitative feel for vectors in various map projections but the vector lengths are not correct. There are also some problems with vectors at region boundaries.

If, instead, we wished to overlay sea level pressure for the South Atlantic only, we would need to take advantage of the `mp_central_meridian` and `mp_standard_parallel` variables. Normally, the map projection scripts calculate the central meridian and standard parallel from the currently set region and generate the `x_page` and `y_page` coordinate transformations accordingly. When we overlay a subregion, we need to rerun the map projection script and pass in values for `mp_central_meridian` and `mp_standard_parallel` so that they are match the previous values and are **not** calculated from the subregion associated with the overlay. (Figure 6_13)

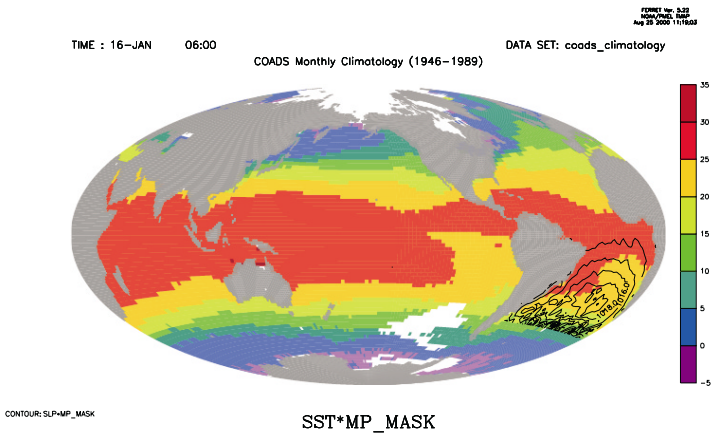



Figure 6_13

```
yes? use coads_climatology
yes? set region/l=1
yes? set grid sst
yes? go mp_hammer
yes? go mp_aspect
yes? shade/noaxis sst*mp_mask, x_page, y_page
yes? go mp_fland
yes? list mp_central_meridian, mp_standard_parallel
        LONGITUDE: 20E to 20E(380)
        LATITUDE: 90S to 90N
 Column  1: MP_CENTRAL_MERIDIAN is (MP_X[I=@MAX] + MP_X[I=@MIN])/2
 Column  2: MP_STANDARD_PARALLEL is (MP_Y[J=@MAX] + MP_Y[J=@MIN])/2
        MP_CENTRMP_STAND
I / *:     200.0  0.0000
yes? go mp_hammer 200 0
yes? set region/x=60w:20e/y=45s:0n
yes? set grid slp
yes? contour/over slp, x_page, y_page
```

**Note:** Had we used `go mp_hammer 200 0` in the beginning we would not have had to rerun `mp_hammer`.

## Ch6 Sec10.4.3. Map Projection scripts

Here is the list of map projection scripts delivered with Ferret. (The techniques used are quite general and can be applied to most map projections.)

| Ferret script | Projection name |
|---|---|
| mp_bonne.jnl | Bonne |
| mp_craster_parabolic.jnl | Craster Parabolic |
| mp_eckert_greifendorff.jnl | Eckert Grifendorff |
| mp_eckert_iii.jnl | Eckert III |
| mp_eckert_v.jnl | Eckert V |
| mp_hammer.jnl | Hammer |
| mp_lambert_cyl.jnl | Lambert Cylindrical Equal Area |
| mp_mcbryde_fpp.jnl | McBryde Flat Polar Parabolic |
| mp_orthographic.jnl | Orthographic |
| mp_plate_caree.jnl | Plate Caree |
| mp_polyconic.jnl | Polyconic |
| mp_sinusoidal.jnl | Sinusoidal |
| mp_stereographic_eq.jnl | Stereographic Equatorial |
| mp_stereographic_south.jnl | Stereographic North |
| mp_vertical_perspective.jnl | Stereographic South |
| mp_vertical_perspective.jnl | Vertical Perspective |
| mp_wagner_vii.jnl | Wagner VII |
| mp_winkel_i.jnl | Winkel I |

Here is the list of utility scripts to support curvilinear coordinates

| Ferret script | Function |
|---|---|
| mp_demo.jnl | demonstration of various map projections |
| mp_fland.jnl | curvilinear version of `fland.jnl` |
| mp_graticule.jnl | creates a graticule (lines of longitude and latitude) over the whole globe or any portion |
| mp_label.jnl | correctly places labels using lat-lon coordinates |
| mp_land.jnl | curvilinear version of `land.jnl` |
| mp_land_stripmap.jnl | creates a land-centric interrupted map using the current projection |
| mp_line.jnl | correctly plots user lat-lon data on the map |
| mp_ocean_stripmap.jnl | creates an ocean-centric interrupted map using the current projection |
| mp_polygon | overlays a “map projected” polygon |

# Chapter 7: HANDLING STRING DATA: "SYMBOLS"

Ferret offers a variety of tools for manipulating strings through the use of "symbols" (variables defined to be strings). The following are the relevant commands:

DEFINE SYMBOL
usage:
DEFINE SYMBOL symbol_name = string

SHOW SYMBOL
usage:
SHOW SYMBOL/ALL
SHOW SYMBOL symbol_name
SHOW SYMBOL partial_name

CANCEL SYMBOL
usage:
CANCEL SYMBOL/ALL
CANCEL SYMBOL symbol_name

Legal symbol names must begin with a letter and contain only letters, digits, underscores, and dollar signs.

To invoke symbol substitution—the replacement of the symbol name with its (text) value—within a Ferret command include the name of the symbol preceded by a dollar sign in parentheses.

For example,

```
yes? DEFINE SYMBOL hi = hello everyone
yes? MESSAGE ($hi)              ! issues "hello everyone" msg
```

It is also possible to nest symbol definitions, as the following commands illustrate:

```
yes? DEFINE SYMBOL label_2 = My test label
yes? DEFINE SYMBOL number = 2
yes? SAY ($label_($number))
      My test label
```

## Ch7 Sec1. AUTOMATICALLY GENERATED SYMBOLS

A number of useful symbols are automatically defined whenever Ferret sets up a plot. Following any plotting command issue the command SHOW SYMBOLS/ALL to see a list. Consult the PLOT PLUS for Ferret Users Guide (section "General Global Symbols") for detailed de-

scriptions of the plot symbols. For example, if we wish to place a label “hello” at the upper right corner of a plot we might do the following

```
yes? PLOT/I=1:100 SIN(I/6)
yes? LABEL/NOUSER ($ppl$xlen) ($ppl$ylen) 1 0 .2 hello
```

This labeling procedure would work regardless of the aspect ratio of the plot. Use the command SHOW SYMBOL/ALL to see the symbols (and see “General Global Symbols” in the PLOT+ Users Guide).

## Ch7 Sec2. USE WITH EMBEDDED EXPRESSIONS

When used together with Ferret embedded expressions symbols can be used to perform arithmetic on the plot geometry. For example, this command will locate the plot title in bold at the center of a plot regardless of the aspect ratio:

```
yes? LABEL/NOUSER `($ppl$xlen)/2` `($ppl$ylen)/2` 0 0 .2 @AC($labtit)
```

## Ch7 Sec3. ORDER OF STRING SUBSTITUTIONS

The above example illustrates that the order in which Ferret performs string substitutions and evaluates immediate mode expressions in the command line is significant. The successful evaluation of the embedded expression `` `($ppl$xlen)/2` `` requires that `($ppl$xlen)` is evaluated before attempting the divide by 2 operation. The order of Ferret string substitutions is as follows:

1. substitute “GO” command arguments of the form “$1”, “$2”, ...
2. substitute symbols of the form ($symbol_name) (discussed here)
3. substitute command aliases
4. substitute immediate mode mathematical expressions

For example, if the script snoopy.jnl contains

```
DEFINE SYMBOL fcn = $1
DEFINE ALIAS ANSWER LIST/NOHEAD/FORMAT=(“Result is ”,$2)
ANSWER `($fcn)(($3^2)/2)`+5
```

then the command

```
yes? GO snoopy EXP F5.2 2.25
```

would evaluate to

```
DEFINE SYMBOL fcn = EXP
DEFINE ALIAS ANSWER LIST/NOHEAD/FORMAT=(“Result is ”,F5.2)
LIST/NOHEAD/FORMAT=(“Result is ”,F5.2) `EXP((2.25^2)/2)`+5
```

and would result in Ferret output of “Result is 17.57.”

## Ch7 Sec4. CUSTOMIZING THE POSITION AND STYLE OF PLOT LABELS

All of the plot labels generated by Ferret are automatically defined as symbols. This includes the title ($labtit), X and Y axis labels ($labx),($laby), as well as the position labels (latitude, longitude, depth, time), which are normally placed at the upper left on a plot (see “Labels,” p. 136). Sometimes it is desirable to change the location, size or fonts of these labels. The symbol facility makes it possible to do this in a way that is independent of the particular label strings or plot aspect ratio. See the demonstration script symbol_demo.jnl for an example.

## Ch7 Sec5. USING SYMBOLS IN COMMAND FILES

Often in Ferret command files the identical argument substitutions must be repeated at several points in the file. Using symbols it is possible to write “cleaner” Ferret scripts in which the argument substitution occurs only once—to define a symbol which is used in place of the argument thereafter. See the demonstration script symbol_demo.jnl for an example.

## Ch7 Sec6. PLOT+ STRING EDITING TOOLS

The PLOT+ program provides a variety of tools for editing symbol strings. See the PLOT+ Users Guide for further information. A sample usage:

```
yes? DEFINE SYMBOL test = my string
yes? PPL SET upper_test $EDIT(test,UPCASE)
yes? SHOW SYMBOL upper_test
UPPER_TEST = "MY STRING"
```

## Ch7 Sec7. SYMBOL EDITING

Symbols may be edited and checked using the same controls that apply to journal file arguments.

The section of this users guide entitled “Arguments to GO tools” (p 20) describes the syntax for checking and editing arguments. The identical syntax applies to symbols. As with the GO tool arguments (e.g., “$4”), all string manipulations are case insensitive.

In brief, the capabilities include:

**default strings**

If a symbol is undefined a default value may be provided using the pattern ($my_symbol%my default string%). For example,

```
($SHAPE%XY%)
```

**check against list of acceptable values**

A list of acceptable string values may be provided using the pattern ($my_symbol%|option 1|option 2|%). For example,

```
($SHAPE%|X|Y|Z|T|%)
```

will ensure that only 1-dimensional shapes (X, Y, Z, or T) are acceptable.

**string substitution**

Any of the optional string matches provided can invoke a substitution using the pattern ($my_symbol%|option 1>replacement|%). For example,

```
($SHAPE%|X>I|Y>J|Z>K|T>L|%)
```

will substitute I for X or J for Y, etc.

**Asterisk (“*”) provides default substitution**

The asterisk character matches any string. For example,

```
($SHAPE%|X|Y|Z|T|*>other%)
```

will always result in “X,” “Y,” “Z,” “T,” or “other.”

**Asterisk (“*”) provides limited string editing**

The asterisk character, when used on the right hand side of a string substitution, inserts the original symbol contents

```
($SHAPE%|*>The shape is *|%)
```

**error message control**

An error message can be provided if the symbol is undefined or doesn’t match any options. The pattern for this is ($my_symbol%|option 1|option 2|<error message text %). For example,

```
($SHAPE%|X|Y|Z|T|<Not a 1-dimensional shape%)
```

## Ch7 Sec8. SPECIAL SYMBOLS

There are a few symbols, generated automatically by plots, which are not documented in the PLOT PLUS for Ferret Users Guide. Those are shown like all symbols by SHOW SYMBOLS, but cannot be redefined by the user.

```
PPL$XPIXEL
PPL$YPIXEL
```

the number of pixels in the horizontal (X) and vertical (Y) size of the current Ferret output window. Note: these are "0" if there is no current window -- hence they can be used as a test of whether there is an open window.

```
BYTEORDER
```

gives "BIG" or "LITTLE" according to endianness of the CPU

```
FERRET_VERSION
```

gives the Ferret version

PPLUS defines a number of global symbols which are available to the user. They are documented in the PPLUS Users Guide, section 7.3, and listed in the chapter “Customizing Plots”, section PPLUS special symbols (p. ).

# Chapter 8: WORKING WITH SPECIAL DATA SETS

## Ch8 Sec1. WHAT IS NON-GRIDDED DATA?

Many data sets which are not normally regarded as "gridded" can nonetheless be managed, analyzed, and visualized effectively in a gridded data framework. Track lines, "point data", etc. are common examples of "non-gridded" data. Profiles and time series, although they are individually simple one-dimensional grids, have a non-gridded structure when considered as a collection, which is often essential.

This chapter addresses a number of classes of non-gridded data sets and offers approaches that make it straightforward to work with these data types in Ferret's gridded data framework. The approaches are all conceived to facilitate a fusion of these data types—so that multiple data types may be easily combined in calculations..

"Point data" refers to collections of values at scattered locations and times. An example would be the column burden of oceanic NO3 and the scattered locations and times at which the measurements were made.

- If at each point of the data scattered there is a vertical profile of values then see COLLECTIONS OF VERTICAL PROFILES (p. 177).
- If at each point of the data scattered there is a time series of values then see COLLECTIONS OF TIME SERIES (p. 180).
- If at each point of the data scattered there is a 2-dimensional grid in the ZT plane then see COLLECTIONS OF TIME SERIES (p. 180).
- If at each point of the data scattered there is a time series of values then see COLLECTIONS OF TIME SERIES (p. 180).

## Ch8 Sec2. POINT DATA

In a gridded context point data is best viewed as a collection of 1-dimensional variables, where the axis of each variable is the index value, 1, 2, 3, ... of the individual point in the scatter. Thus, continuing our example of an oceanic NO3 data set, we would want to view this as four variables, longitude, latitude, date, and burden, where each variable was defined on a one-dimensional axis of earthquake number. Typically, this sort of data is organized in a table of the form

| Index | longitude | latitude | year | month | day | N03 |
|---|---|---|---|---|---|---|
| 1 | 160 | 30 | 1968 | 11 | -999 | 6.2 |
| 2 | 33.1 | 60.2 | 1992 | 5 | 13 | 5.5 |
| ... | | | | | | |

## Ch8 Sec2.1. Getting point data into Ferret

Since point data sets are most commonly available in table form, where the columns of the table are the variables and each row of the table is a separate point. In the chapter "Data Set Basics", section "Reading ASCII Files" (p. 37), example 2 and subsequent examples show how such a file might be read into Ferret.

For example, let us suppose that the file above is introduced to Ferret with the command

```
yes? FILE/VAR="index,lon,lat,yr,mn,day,NO3"/SKIP=1 my_data_file.dat
yes? SHOW DATA my_data_file.dat

         currently SET data sets:
        1> ./my_data_file.dat  (default)
     name     title                          I          J          K
L
     LON      LON                         1:20480    ...        ...
…
     LAT      LAT                         1:20480    ...        ...
…
     YR       YR                          1:20480    ...        ...
…
     MN       MN                          1:20480    ...        ...
…
     DAY      DAY                         1:20480    ...        ...
…
     NO3      NO3                         1:20480    ...        ...
…
```

Note that the SET VARIABLE command would normally be used as well to assign titles, units, and missing value flags to the variables.

Also note that until the first data is actually requested from the file, Ferret does not know the size of the file. The /GRID= option may be used to tell Ferret what size to expect. Lacking a /GRID specification the "1:20480" is the size of the default grid "EZ." After the first data access SHOW GRID will reveal the true size of the file, instead. If the size still appears to be 20480 it may be that the default grid EZ was not large enough, and the /GRID qualifier must be used to pre-allocate sufficient space.

## Ch8 Sec2.2. How point data is structured in Ferret

In table form (above) each column represents a dependent variable; the column for "burden" and the column for "latitude" have equal status. In many cases this is an adequate representation. For example, a plot of NO3 burden versus latitude could be produced with the command

```
yes? PLOT/VS lat, NO3
```

To combine point data organized in tables with gridded data sources, say a gridded field of oceanic temperature two approaches are available. Either the gridded data may be viewed in the structure of the table, or the scattered data may be viewed in a geo-referenced 1-dimensional

grid structure. The problem to be solved determines which approach is suitable. The next two sections describe these two approaches.

## Ch8 Sec2.2.1. Working with dates

Ferret V5.0 does not understand formatted dates inside of generic data ASCII files. To use the dates intelligibly inside of Ferret you

1. Need to get the year, month, and day fields broken out separately or provide a Julian day.
2. Can create a Julian date from year, month, day using function DAYS1900. If a time origin other that 1-jan-1900 is needed subtract DAYS1900(year0, mon0, day0)
3. Can create an axis of dates as done in the preceding latitude axis example.

See the chapter "Grids and Regions", section "Time" (p. 119) and the section in the chapter "Converting to NetCDF" on "Converting time word data to numerical data" (p. 206) for details of creating time axes.

## Ch8 Sec2.3. Subsampling gridded fields onto point locations and times

Ferret can be used as a tool to extract variables from gridded data sets at time/space locations to match the scatter of the point data. In this form they may, effectively, be combined into the table of data read from the ASCII (or binary) file. For example, suppose we want to obtain values of sea surface temperature at the locations of our NO3 samples, from a climatological annual average SST field. This may be accomplished simply with

```
yes? use coads climatology
yes? let ssttav = sst[l=1:12@ave]
yes? let my_lon = lon[d=my_data_file.dat]
yes? let my_lat = lat[d=my_data_file.dat]
yes? LET sst_xy = SAMPLEXY(ssttav, my_lon, my_lat)
```

Suppose that, instead our SST variable was a monthly climatology field. The variable sst_xy as defined above would then have a two-dimensional structure: sample index by 12 months. To sample in time, as well, we use

```
yes? LET sst_t = SAMPLET_DATE(sst_xy,0,mn,day,0,0,0)
```

Note that the year was entered simply as 0, since SST is a climatological variable.

In this example we sampled a field in X, Y, and T. The sst data was sampled at each time. If we were sampling a field which had a Z axis, that axis would be inherited from the first argument to SAMPLEXY in the same way; it would be sampled at the (x,y) points at each Z level.

## Ch8 Sec2.4. Defining gridded variables from point data

For some calculations one may want to let Ferret know which of the variables are dependent (measurements) and which are independent (coordinates). For example, suppose we wish to compute the average column burden of NO3 as a function of latitude. Burden here is an integral of the concentration NO3 over depth. We will want to see our variable burden on an axis of latitude.

The steps to do this are

1. In general, the latitude variable will not be sorted into strictly increasing order — needed to create an axis. Determine the sorting order for latitude using
   ```
   yes?LET lat_index = SORTI(lat)
   ```
2. Create a latitude grid
   ```
   yes? DEFINE AXIS/FROM/NAME=lat_ax/Y/UNITS=degrees SAMPLEI(lat,
lat_index)
   yes? DEFINE GRID/Y=lat_ax glat
   yes? LET NEW = Y[g=glat] ! a dummy variable to use in RESHAPE below
   ```
3. Define your function for the burden based on the variable NO3, on the command line or using your script my_brdn.jnl.
   ```
   yes? GO my_brdn NO3 burden
   ```
4. Define a new variable burden_on_lat using this axis
   ```
   yes? LET sorted_burden = SAMPLEI(burden, lat_index)
   yes? LET burden_on_lat = RESHAPE( sorted_burden, new )
   ```
5. Now, to plot the NO3 burden averaged into 5 degree latitude bands we could use
   ```
   yes? PLOT burden_on_lat[Y=60s:30n:5@AVE]
   ```

## Ch8 Sec2.5. Visualization techniques for point data

Scattered point data can be displayed in a number of ways.

A simple scatter plot showing the locations of points

```
yes? PLOT/VS lon,lat
yes? GO land
```

Use `GO/help` land for an explanation of resolving incompatible longitude encodings, should they arise.

A scatter plot in which the symbols are colored by value with control over the color palette and resolution can be made using the polymark.jnl script. For example, to plot using stars symbols in color levels by 10s use

```
yes? GO polymark POLYGON/LEV=(0,100,10) lon lat NO3 star.
```

Type `GO/HELP` polymark for more options.

See also the chapter "Customizing Plots", section "Map Projections" (p. 160) for guidance on plotting scattered data. The map projection scripts can be used in conjunction with the above.

## Ch8 Sec3. VERTICAL PROFILES

A single profile, possibly consisting of multiple variables, can be regarded as a simple 1-dimensional data set. Ferret's plotting and analysis tools apply in a straightforward manner.

Collections of profiles resemble point data sets in their X,Y, and T structure, however at each point there is a 1-dimensional Z-axis structure. In general, the Z axes at each point may differ.

## Ch8 Sec3.1. How collections of profiles are structured in Ferret

If the collection of profiles is sufficiently small (say 4 or fewer) then it is straightforward to handle them simply as 4 separate data sets. The D= qualifier may be used to designate which profile is being referred to. The IF ... THEN ... ELSE syntax may be used to combine the profiles into expressions.

As the number of profiles in the collection grows larger, however, it becomes necessary to merge them into a single structure. Typically, the sequence number of the profile, 1, 2, ...,N, becomes the X axis of the collection. The longitude, latitude, and time of each profile become dependent variables indexed by the sequence number. The Z structures of the profiles are blended into a single Z axis by a choice of techniques. The steps to creating a blended data set then become:

1. Determine the nature of the Z axis to be used and the collection of variables to be defined on the grid
2. Create an empty grid with the desired structure in a file
3. Populate the file with the profiles, each profile in turn.

The determination of the Z axis structure may be by any of these techniques:

1. Supply an arbitrary Z axis to which all of the individual profiles will be regridded by linear interpolation. This technique produces a data set which is very easy to work with and small in size, however, some of the data have been altered by linear interpolation. The default Ferret regridding (GZ=@LIN) is used for this technique.
2. Create a Z axis which is a superset of the Z axis points from all of the grids. In the final data set this axis will be sparsely populated, containing only those Z points that were actually present in each profile.
3. This technique produces a data set which is 100% faithful to the original data and reasonably easy to work with, but may become very large if the number of profiles is large and the Z axes vary greatly. Ferret "exact match" regridding (GZ=@XACT) is used for this technique.

4. Do not create a Z axis at all — instead store the Z coordinates as a dependent variable. The Z axis becomes simply an index counter of length equal to the longest profile. This technique produces a data set which is 100% faithful to the original data and of modest size, however it is the most laborious to work with.

The choice of technique depends on the nature of the profile collection and the types of analysis or visualization to be done. Often it is desirable to combine technique 1, which is fast and simple with 2 or 3, which can be used for spot checking if there is a question of data fidelity. If method 3 is chosen (Z coordinates in a dependent variable) the techniques for handling the variables are very similar to sigma coordinate data, described in a separate section of this chapter (p. 181).

## Ch8 Sec3.2. Getting profile data into Ferret

As of 4/99 the approaches to merging collections of profiles into a single structure are still "manual." (Data which are stored as global attributes in the input files, as is done in EPIC files, are lost in this process.) This text describes an example of the manual process used, where the target Z axis is created arbitrarily and data are interpolated to it. In this example the profiles are read from ASCII files, so the Z axis of each profile has to be created. This example does not save the longitude, latitude, and time positions of the casts.

```
! for this example we begin by manufacturing some data
! ... pretend this is one of your casts - unequal vertical spacing
list/file=test_cast.dat/nohead/form=(2f)/i=1:10 10*i+randu(i), sin(i/6)

! create a grid suitable for ALL casts together
! make the points regular in X and Z ... they need not be, however
define axis/depth/z=0:1000:20/unit=meters zall ! ARBITRARTY Z AXIS
define axis/x=0:9:1/unit="sequence" xall
define grid/x=xall/z=zall gall

! create an empty output file
! if we were reading netCDF files we would create variables to hold
! longitude, latitude, and time (year, month, day).
! A latitude output variable, for example, is created below
let outvar = 1/0 * x[g=gall] * z[g=gall]
set variable/title="My merged var"/units="my units" outvar
save/file=all_casts.cdf/ilimits=1:10/zlimits=0:1000 outvar
let lat = 1/0*X[gx=gall]
set variable/title="Latitude"/Units="degrees" lat
save/append/file=all_casts.cdf/ilimits=1:10 lat

! read in a single cast (the fake data we created)
! if we were reading a NetCDF file this block would be unnecessary
file/var=depth,invar test_cast.dat
define axis/from_data/z/depth/name=z1cast/unit=meters depth    ! make Z
                                                    ! axis for 1
profile
define axis/x=0:0:1/unit="sequence" x1cast     ! sequence no. of first
cast
define grid/x=x1cast/z=z1cast g1cast
canc data 1

! save first cast interpolated to many-point Z axis
```

```
file/var="-,invar"/grid=g1cast test_cast.dat
let outvar = invar[g=gall]
save/append/file=all_casts.cdf outvar[I=1]
canc data 1
! if available, output latitude thusly
!   LET lat = 0*X[g=gall] + RESHAPE(Y[G=invar],X[gx=gall])
!   SAVE/append/file=all_casts.cdf lat[I=1]

! save next cast
define axis/x=1:1:1/unit="sequence" x1cast      ! X position of 2nd
cast
file/var="-,invar"/grid=g1cast test_cast2.dat
save/append/file=all_casts.cdf outvar[I=2]
canc data 1

! etc for next 8 casts …
! This may be automated with:   REPEAT/I=1:10 GO output_one_profile
! where the script output_one_profile.jnl reads profile file names from
a list
```

The output data set which we create will be structured as follows:

```
yes? use all_casts
yes? show data
     currently SET data sets:
    1> ./all_casts.cdf  (default)
 name     title                                I          J          K
L
 OUTVAR   My merged var                      1:10       ...        1:51
…
 LAT      Latitude                           1:10       ...        ...
…
```

## Ch8 Sec3.3. Defining vertical sections from profiles

In the data set created above the profiles may or may not be ordered as needed to create a valid section. There are many possible ways to order the data. Often more than one technique is applicable to a single data set. The data may be ordered along a ship track, ordered by increasing latitude, ordered by path distance along a regression line, etc.

Continuing with the example above, we can order the profiles into increasing latitude with:

```
yes? let order = SORTI(lat)
yes? let section = SAMPLEI(outvar, order)
```

Other definitions of the variable order may be created by straightforward means to apply other ordering principles.

As defined above, “section” has an X axis which is the values 1, 2, 3,...N from the Ferret ABSTRACT axis. To cast this on a proper latitude axis, use these two steps:

```
yes? DEFINE AXIS/Y/NAME=yax_sect/FROM_DATA/UNITS=degrees SAMPLEI(lat,
order)
yes? LET ysection = RESHAPE(section,Y[gy=yax_sect]+Z[gz=all])
```

## Ch8 Sec3.4. Visualization and analysis techniques for profile sections

The variables “section” and “ysection” defined above may be plotted and analyzed with the normal gridded plot commands. For examples,

```
yes? CONTOUR section  ! contour plot ordered on X=1,2,3,...
yes? FILL ysection    ! color contour plot on formatted latitude axis
yes? PLOT/Y=20S/Z=100:500 ysection ! profile at 20 south
yes? PLOT ysection[Z=@loc:20]      ! depth of 20 degree isotherm
```

## Ch8 Sec3.5. Subsampling gridded fields onto profile coordinates

The technique described for sampling grids at scattered point values will work unmodified for collections of vertical profiles. The Z coordinate of the gridded variable will be retained unmodified throughout the sampling operations. Regrid the final result variable to other Z axes as desired.

## Ch8 Sec4. COLLECTIONS OF TIME SERIES

Handling of collections of time series is analogous to handling collections of vertical profiles, described above. The choices of

1. a single interpolated time axis (using the default, GT=@LIN, regridding)
2. a super-set of all times axis (using “exact match,” GT=@XACT, regridding)

should be considered. Choice 3, in which time would be handled as an independent variable, is possible, but awkward, due to the multiplicity of time encodings.

## Ch8 Sec5. COLLECTIONS OF 2-DIMENSIONAL GRIDS

Handling collections of 2-dimensional grids (e.g. ZT grids from acoustic current profilers) is a straightforward extension of the techniques described under collections of profiles. If the time axes of the input grids are all identical, no additional work is needed beyond the techniques described there. If the time axes differ then follow the guidance given under Collections of Time Series, using intermediate variable definitions that reconcile the time axes into a single uniform axis before saving the input variables into a merged output file.

## Ch8 Sec6. LAGRANGIAN DATA

Lagrangian data (ship tracks, drifters, etc.) is a special case of scattered point data described in a preceding section. In the terminology of "Defining gridded variables from point data" Lagrangian data is simply point data organized onto a 1-dimensional time axis grid.

## Ch8 Sec6.1. Visualization techniques for Lagrangian data

Ferret has several visualization tools that specifically address the needs of Lagrangian data. There are three scripts:

| | |
|---|---|
| polymark (polymark_demo) | marks value-colored symbol at each location |
| polytube (polytube_demo) | creates a line following the Lagrangian track with color varying according to a Lagrangian variable |
| trackplot (trackplot_demo) | creates a line plot of a Lagrangian variable where the zero line of the plot follows the Lagrangian track |

Overlays of the trackplot script are useful to visualize more than one variable. Run the demonstration scripts noted above for each tool for an example of its use with Lagrangian data.

## Ch8 Sec7. SIGMA COORDINATE DATA

With sigma coordinate data the vertical coordinate (or layer thickness) is available as a dependent variable and the Z axis of the sigma-encoded variables is layer number (the Z index). This is precisely analogous to method 3 of handling collections of profiles, above.

See also the FAQ on Using Sigma Coordinates.

## Ch8 Sec7.1. Visualization techniques for sigma coordinate data

Visualizations of sigma coordinate data in vertical section planes are best handled with the 3-argument versions of the SHADE, FILL, CONTOUR and VECTOR commands. See further information in Customizing Plots (, p. 129).

For visualization of sigma coordinate data in other planes or orientations use the techniques described in the next section.

## Ch8 Sec7.2. Analysis techniques for sigma coordinate data

Analysis of sigma coordinate data, which requires shifting to depth or pressure coordinates, is facilitated by the function ZAXREPLACE, which converts from layer number to other vertical coordinate axes. See sigma_coordinate_demo.jnl for an example. If the data set provides layer thickness rather than depth a depth variable may be created using integration with @iin.

## Ch8 Sec8. CURVILINEAR COORDINATE DATA

By "curvilinear coordinate data" we refer to data which is curvilinear in the XY plane there. We presume that the X,Y coordinates (typically longitude, latitude) are available through other dependent variables.

## Ch8 Sec8.1. Visualization techniques for curvilinear coordinate data

Visualizations of curvilinear coordinate data in the XY plane section planes are best handled with the 3-argument versions of the SHADE, FILL, and Contour commands. See further information in the chapter "Customizing Plots" (p. 129).

For visualization of curvilinear coordinate data in other planes or orientations use the techniques described under "Analysis techniques for curvilinear coordinate data."

## Ch8 Sec8.2. Analysis techniques for curvilinear coordinate data

Analysis of curvilinear coordinate data may be done in the curvilinear coordinate system or in a rectilinear (including lat-long) coordinate system. If the analysis is done in the curvilinear coordinate system, it is the responsibility of the user to ensure that the proper geometric factor are applied when integrals and derivatives are computed. Converting other fields to the curvilinear coordinate system is most easily accomplished with the function SAMPLEXY.

To perform the analysis in a rectilinear coordinate system, the conversion of the curvilinear data is most easily done with SAMPLEXY_CURV (under development—6/00 ).

## Ch8 Sec9. POLYGONAL DATA

By "polygonal data" we refer to a class of point data set where each point represents a polygonal region rather than a single coordinate. An example of polygonal data would be a value associated with each state in the United States.

## Ch8 Sec9.1. Visualization techniques for polygonal data

Visualizations of polygonal data is best handled with the POLYGON command. If the coordinates of the polygon vertices are available in 2-dimensional arrays, XPOLY and YPOLY, in which the axes of the arrays are the polygon vertices and the sequence of polygons the use of the POLYGON command is straightforward. The POLYGON command can also handle sequences of polygons encoded in 1-dimensional arrays with missing values separating each polygon.

## Ch8 Sec9.2. Analysis techniques for polygonal data

Ferret version 5.0 does not have any tools specifically addressing the analysis of polygonal data sets. The analysis of these data sets in Ferret requires the creation of a gridded mask field corresponding to the polygonal regions (an external function could be written that would create a gridded mask of arbitrary resolution from polygonal coordinates.)

Once the mask is created, the standard gridded operators for averaging, integrating, etc. can be used. For example, if variable cal_mask contains a gridded mask of the state of California on latitude and longitude axes of 10 minute resolution then this definition would compute the average of a gridded variable, var, over California:

```
yes? let cal_var = mask * var[g=mask]
yes? let cal_average = cal_var[x=@ave, y=@ave]
```

# Chapter 9: COMPUTING ENVIRONMENT

## Ch9 Sec1. SETTING UP AN ACCOUNT

This discussion assumes that Ferret is already installed on your system. Installation documentation is available separately from the Ferret Downloads web page

STEP 1

Execute interactively or add to your .login file the Unix C-shell command

```
% source /usr/local/ferret_paths
```

(**Note:** If this command doesn't work consult your system manager, who may have placed ferret_paths in a different directory.)

The Ferret program requires access to several files and directories. These Unix paths are stored in environment variables defined by the file "ferret_paths". Your Unix account must be "made aware" of where the Ferret utilities are located. This is done by adding to the definition of your environment variable PATH the directory "$FER_DIR/bin". Unless your system manager has modified the typical setup, this will occur automatically when you execute the above command.

STEP 2 (personal customization—optional)

Execute the "cp" command below:

```
% cp $FER_DIR/bin/my_ferret_paths_template \
```

$HOME/my_ferret_paths

Then use a text editor to customize my_ferret_paths. Instructions are inside the file.

Some of the Ferret environment variables identify files and directories that are integral to the Ferret program, but others identify files that you may maintain—your data files, GO scripts, and palette files, for example. (The environment variables that you may want to customize are discussed at the end of this section.) To assist in customizing the Ferret environment variables the template file in the "cp" command, above, has been provided. The file is self-explanatory.

STEP 3

Execute the command below interactively or add it to your .login file.

```
% setenv DISPLAY node:0.0
```
e.g., `% setenv DISPLAY anorak:0.0`

This command sets the environment variable "DISPLAY" to point to the workstation console or X-terminal where you want Ferret graphical output displayed. In the example above, graphical output is directed to the screen of workstation "anorak."

## Ch9 Sec2. FILES AND ENVIRONMENT VARIABLES USED BY FERRET

.ferret—the Ferret initialization file. This optional file holds a list of Ferret commands that will be executed immediately each time Ferret is started, permitting Ferret to be tailored to individual needs and styles. The file must be located in your $HOME (login) directory. A simple way to set up such a file is to enter Ferret, enter the commands that you want executed each time you enter Ferret, exit Ferret and rename the file "ferret.jnl" to ".ferret". Thereafter, all commands in ".ferret" will be executed automatically whenever you enter Ferret.

The following environment variables are defined in the file ferret_paths:

FER_DATA—a list of directories to be searched to locate data files. Usually this list includes ".", the current directory, and $FER_DSETS/data, a directory of sample data sets provided with Ferret. Your system manager may have set this variable to include other data areas as well. This is the list of directories searched to locate NetCDF files.

FER_DESCR—a list of directories to be searched to locate descriptor files. Descriptors are required by Ferret to access data sets that are in Ferret's "GT" (grids at timesteps) or "TS" (time series) formats. Usually this list includes ".", the current directory, and $FER_DSETS/descr, a directory of sample descriptors provided with Ferret.

FER_GRIDS—a list of directories to be searched to locate grid definition files. Data sets will usually have a grid definition file associated with them so that the grids on which the data are defined may be known.

FER_DIR—top directory of the Ferret distribution on your system.

FER_DSETS—directory of sample data sets provided with the Ferret distribution.

FER_PALETTE—a list of directories to be searched to locate palette files. Usually this list includes "." and $FER_DIR/ppl.

FER_GO—a list of directories to be searched to locate GO scripts. This list usually includes ".", $FER_DIR/go, $FER_DIR/examples (demonstrations and tutorial), and $FER_DIR/contrib (user contributions demonstrating various applications; accuracy not guaranteed).

FER_EXTERNAL_FUNCTIONS—a list of directories to be searched to locate the shared object files (.so files) for external functions. By default this list includes the location of the example functions and the functions included with the Ferret distribution.

## Ch9 Sec3. MEMORY USE

Ferret indicates memory problems by issuing the error message "insufficient memory." If memory is a problem running Ferret the following suggestions may help:

1) Use the command SET MEMORY/SIZE=nnn to increase the memory cache region available to Ferret.

2) Use the command SET MODE DESPERATE to determine the threshold size of memory objects at which Ferret will break a large calculation into fragments. A smaller argument value will induce stricter memory management but at a penalty in performance.

3) Use CANCEL MEMORY whenever you are sure that the data referenced thus far by Ferret will not be referenced again. This is particularly appropriate to batch procedures that use Ferret. This eliminates any memory fragmentation that may be left by previous commands.

4) Use CANCEL MODE SEGMENTS to minimize the memory usage by graphics (on a few X-window systems this may prevent windows from being restored after they are obscured).

5) When using DEFINE VARIABLE (alias LET) avoid embedding upper and lower axis bounds within the variable definition. Ferret cannot split up large calculations along axes when the limits are fixed in the definition. For example,

```
yes? LET V2=TEMP/10
yes? PLOT/K=1:10 V2
```

is preferable to

```
yes? LET V2=TEMP[K=1:10]/10
yes? PLOT V2
```

6) Try to group together calculations that are on smaller dimensioned objects. For example, the expression VAR[i=1:100, j=1:100]*2*PI will make less efficient use of cpu and memory than the expression VAR[i=1:100, j=1:100]*(2*PI). The former multiplies each of the 10000 points of VAR by 2 and then performs a second multiplication of the 10000 result points by PI. The latter computes the scalar 2*PI and uses it only once in multiplying the 10000 points of VAR.

7) If one has SET MODE STUPID:weak_cache, then make sure that the region is fully defined (i.e., check SHOW REGION and check the region qualifiers of your command). When the

region along some axis is not specified Ferret defaults to the full span of the data along that axis and is unable to optimize memory usage.

# Ch9 Sec4. HARD COPY AND METAFILE TRANSLATION

## Ch9 Sec4.1. Hard copy

To obtain hard copy of plots produced by Ferret, follow these steps:

1) Within Ferret, enter the command

```
yes? SET MODE METAFILE
```

This tells Ferret to generate a graphic metafile (ANSI/ISO GKSM format) for each plot created thereafter. To stop making the metafiles type

```
yes? CANCEL MODE METAFILE
```

2) Produce each plot as you would normally. Each new plot on your screen generates an additional file named "metafile.plt.~n~" where "n" will be incremented for each metafile. Overlay commands do not produce additional metafiles. (The metafile name may be set by the SET MODE METAFILE command.)

3) After exiting from Ferret use the command Fprint.

> **Note:** If it is necessary to use Fprint without exiting Ferret, then issue the command `yes? PPL CLSPLT`. This will close the current metafile. Note that neither overlays nor additional viewports can be added to the plot after the metafile has been closed.

Fprint is a script which translates metafiles generated by Ferret. It uses the program "gksm2ps" and is intended to simplify sending plots to printers, to an output file only, or to a workstation screen.

For monochrome printers the metafile translator, gskm2ps, uses different line styles (dash-dot patterns) rather than colors for different lines. See Appendix I of the Plotplus manual: Enhancements to Plotplus for a complete list of line styles for monochrome devices.

The Fprint script translates metafiles to Encapsulated PostScript or X-window output. Your system manager should customize the script at your site to permit your specification of the actual printers you have as output devices. Fprint uses standard Unix command line syntax.

```
Fprint [-h] [-P printer || -o file_name || -X]
```

[-p orient] [-# n] [-l line] [-R] metafile(s)

**Options**

| | |
|---|---|
| -h | displays help on your terminal. |
| -P printer | Routes output to named printer. Files will not be renamed by previewing. You will be prompted, however, with an option to delete each metafile after previewing. The output window size will be equivalent to the default size in Ferret (SET WINDOW/SIZE=0.7). |
| -o file_name | Routes output to named disk postscript file. |
| -X | Routes output to your workstation screen. Files will not be renamed by previewing. You will be prompted, however, with an option to delete each metafile after previewing. The output window size will be equivalent to the default size in Ferret (SET WINDOW/SIZE=0.7). |
| -p orient | The page orientation option determines whether the plot will be placed on the page in landscape format, with the horizontal side longer than the vertical, or portrait, with the vertical side longer. Valid option values are “landscape” and “portrait”. The default behavior is to orient the plot to best fit the page. |
| -# n | Specifies number of copies (n). |
| -l line | This option lets you specify line styles. Valid options are “ps” and “cps”. “ps” uses dot-dashed line types; “cps” uses colored lines. The default is “ps” for monochrome printers and “cps” for color printers. |
| -R | Turns off the default behavior of the metafile translator to append a date stamp to metafile names when they are sent to a printer or a disk file. The default action is intended to distinguish metafiles that have been printed out; this option keeps the metafile names unmodified. |

Examples

```
% Fprint metafile.plt
```

renders “metafile.plt” on the default printer identified by the environment variable PRINTER.

```
% Fprint -P myprinter -R metafile.plt*
```

renders all versions of “metafile.plt” on printer myprinter. Does not date stamp them.

```
% Fprint -o my_plot.ps metafile.plt.~1~
```

writes plot “metafile.plt.~1~” to a postscript file named “my_plot.ps”.

## Ch9 Sec4.2. Metafile translation

The command "gksm2ps" allows you to control the translation of the device-independent metafiles made by Ferret into device-specific output files. "gksm2ps" was written by Larry Oolman at the University of Wyoming and modified at NOAA/PMEL for use with Ferret. The "gksm2ps" command uses standard Unix command line syntax. See usage hints provided by the -h option.

```
gksm2ps [-h] [-p landscape||portrait] [-l ps||cps] [-d cps||phaser] \
```

[-X || -o <ps_output_file>] [-R] [-a] [-g WxH+X+Y] file(s)

**Options**

| | |
|---|---|
| -h | prints help message. |
| -p orient | The page orientation option determines whether the plot will be placed on the page in landscape format, with the horizontal side longer than the vertical, or portrait, with the vertical side longer. The default is to orient the plot to best fit the page. |
| -l line | This option permits specification of line styles in the hardcopy plot. Valid options are "ps" (the default) and "cps". "ps" renders lines as solid and dot-dashed and is suited for monochrome printers. "cps" renders lines in color. |
| -d devtype | The target device type of the translator. If the -d option is omitted and output is to a file gksm2ps will use devtype "ps".<br>Valid devtype values:<br>Cps – color PostScript<br>phaser – Tektronix Phaser PX. The phaser is a PostScript printer, but it uses transfer sheets that reduce the usable page size. |
| -X | Sends the output to your X-window for preview. |
| -o ofile | The output will be directed to the file "ofile." Omit both this and the device type option when directing output to your workstation screen with -X. If neither -o nor -X is specified, gksm2ps creates a postscript file in the current directory called "gksm2ps_output.ps". |
| -a | Makes the plot the size of the original plot as specified in PPLUS inches (absolute size), rather than fitting the plot to the page (the default behavior). |
| -g WxH+X+Y | The -g option (-g WxH+X+Y) provides detailed control over the size, position, and aspect ratio of the plot on the printed page. The arguments W, H, X, and Y are given in units of points (1/72 of an inch).<br>Normally when using this option you will want to specify an identical value for both W and H—the size (in points) you want the longer dimension of the plot to be. Unequal values of W and H will alter the aspect ratio of the plot relative to its appearance on your workstation screen. |

**Options**

The X and Y values are the offset of the lower left corner of the plot from the lower left corner of the page. If you want your plot's longer side to be 5 inches long, 3 inches right from the corner, and 2 inches up, for example, specify

```
> lpr my_plot.ps
```

-R Turns off the default behavior of the metafile translator to append a date stamp to metafile names when they are sent to a printer or a disk file. The default action is intended to distinguish metafiles that have been printed out; this option keeps the metafile names unmodified.

If the user does not specify an output option (-o or -X) gksm2ps translates the metafile and produces a PostScript file called gksm2ps_output.ps. After translation by gksm2ps, metafiles are renamed with a date stamp unless -R was specified. To get hard copy printed, the output PostScript file needs to be sent to the appropriate printer.

## Ch9 Sec5. OUTPUT FILE NAMING

Ferret uses a file naming scheme to differentiate successive graphic metafiles and journal files. The scheme is styled after the gnu (Free Software Foundation) emacs editor. The scheme appends numbers to the end of the file name as in the following examples:

```
Metafile.plt.~2~
metafile.plt.~12~
metafile.plt
```

The third example, "metafile.plt" with no suffix appended, is the most recent file. When the next successive file is created, this file will have the suffix ".~nnn~" appended to its name. "nnn" is the current highest file suffix number plus one.

Two Unix tools are provided to assist with managing multiple file suffix numbers:

| | |
|---|---|
| Fpurge | removes all but the current version of the named file (that is, all but the most recent).<br>Example: `% Fpurge ferret.jnl` |
| Fsort | sorts the versions of a file into increasing numerical order<br>Example: `% Fprint 'Fsort metafile.plt*'` |

See the introductory chapter, section "Unix tools," p. 24, for further information.

## Ch9 Sec6. INPUT FILE NAMING

There are several Ferret commands that use filenames. These include:

```
GO filename
SET DATA filename
LIST/FILE=filename  (do not use relative versions (below) with LIST)
USER/FILE=filename
SET MODE META filename
SET MODE JOURNAL filename
SET MODE PPLLIST filename
```

The filename specified can be just the filename itself, or it can include the path to the file. For example:

```
GO ferret.jnl
```
or
```
GO "/home/disk1/jnl_files/far_side.jnl"
```

Note that if the path is specified as part of the filename, the entire name must be enclosed in quotation marks.

## Ch9 Sec6.1. Relative version numbers

Under some circumstances (see the GO command, p. 266) a special syntax called "relative version numbers" will apply. If a filename has a version value of zero or less its value is interpreted relative to the current highest version number.

For example, if the current directory contains the files

```
ferret.jnl   ferret.jnl.~1~   ferret.jnl.~2~   ...  ferret.jnl.~99~
```

then the filename `ferret.jnl.~0~` refers to `ferret.jnl` and the filename `ferret.jnl.~-1~` refers to `ferret.jnl.~99~`.

The syntax for relative version numbers is quite flexible. For example, if the desired file is `ferret.jnl.~99~`, both of the following are valid:

```
yes? GO ferret.jnl.~-1~
```
or
```
yes? GO ferret.jnl~-1
```

# Chapter 10: CONVERTING TO NetCDF

## Ch10 Sec1. OVERVIEW

The Network Common Data Format (NetCDF) is an interface to a library of data access routines for storing and retrieving scientific data. NetCDF allows the creation of data sets that are self-describing and network-transparent. NetCDF was created under contract with the Division of Atmospheric Sciences of the National Scientific Foundation and is available from the Unidata Program Center in Boulder, Colorado (on Internet: unidata.ucar.edu).

This chapter provides directions for creating NetCDF data files. In addition to the documentation provided here, refer to the NetCDF User's Guide, published by Unidata Program Center, for further guidance. A user who uses and creates NetCDF files within the Ferret environment needs no additional software.

NetCDF is a very flexible standard. In most cases there are multiple styles or profiles that could be used to encode data into NetCDF. To resolve the ambiguities inherent in this multiplicity communities of users have banded together to develop profiles—documents that provide more detail on how data should be encoded into NetCDF. Ferret adheres to the COARDS standard. The full standard is available through the Ferret home page on the World Wide Web,

http://ferret.wrc.noaa.gov/noaa_coop/ coop_cdf_profile.html

## Ch10 Sec2. SIMPLE CONVERSIONS USING FERRET

In straightforward conversion operations where ASCII or unformatted binary data files are already readable by Ferret, the conversion to direct access, self-describing NetCDF formatted data can be accomplished by Ferret itself. The following set of examples illustrates these procedures:

**Example 1**

Consider an ASCII file uv.data, with two variables, u and v, defined on a grid 360 by 180. The following set of commands will properly read in u and v and convert them to a NetCDF formatted data set:

```
yes? DEFINE AXIS/x=1:360:1/units=degrees xaxis
yes? DEFINE AXIS/y=1:180:1/units=degrees yaxis
yes? DEFINE GRID/x=xaxis/y=yaxis uv_grid
yes? FILE/GRID=uv_grid/BAD=-999/VAR="u,v"  uv.data
yes? SET VARIABLE/TITLE="zonal velocity" u
yes? SAVE/FILE=uv.cdf u,v
```

See command DEFINE AXIS in the Commands Reference (p. 252). See the chapter "Grids and Regions" (p. 101) for setting up formatted latitude, longitude and time axes.

**Example 2**

Consider now two separate ASCII files, u.data and v.data, defined on a grid 360 by 180. The following set of commands will properly read in u and v and convert them to a single NetCDF formatted data set:

```
yes? DEF AXIS/x=1:360:1/units=degrees xaxis
yes? DEF AXIS/y=1:180:1/units=degrees yaxis
yes? DEF GRID/x=xaxis/y=yaxis uv_grid
yes? FILE/GRID=uv_grid/BAD=-999/VAR=u u.data
yes? FILE/GRID=uv_grid/BAD=-999/VAR=v v.data
yes? SAVE/FILE=uv2.cdf u[D=1]
yes? SAVE/APPEND/FILE=uv2.cdf v[D=2]
```

**Example 3—multiple time steps**

Consider 12 ASCII files, uv.data1 to uv.data12, each defined on the same grid as above but each representing a successive time step. The following set of commands illustrates how to save these data into a single NetCDF data set (time series):

```
yes? DEF AXIS/x=1:360:1 xaxis
yes? DEF AXIS/y=1:180:1 yaxis
yes? DEF AXIS/t=1:1:1 taxis1
yes? DEF GRID/x=xaxis/y=yaxis/t=taxis1 uv_grid1
yes? FILE/GRID=uv_grid1/BAD=-999/VAR="u,v" uv.data1
yes? SAVE/FILE=uv1_12t.cdf u,v
yes? CANCEL DATA uv.data1
yes? DEF AXIS/t=2:2:1 taxis1
yes? FILE/GRID=uv_grid1/BAD=-999/VAR="u,v" uv.data2
yes? SAVE/APPEND/FILE=uv1_12t.cdf u,v
. . .
```

and so on, redefining the time axis to be 3:3:1, 4:4:1, ... each time a new file is set.

**Example 4—multiple slabs**

The procedure used in example 3, above, is possible because NetCDF files can be extended along the time axis. In order to append multiple levels (Z axis), the NetCDF file must first be created including all of its vertical levels (the levels initially are filled with a missing data flag).

Consider 5 ASCII files, uv.data1 to uv.data5, each defined on the same grid as above but each representing a successive vertical level. The following set of commands illustrates how to save these data into a single NetCDF data set:

```
yes? DEF AXIS/x=1:360:1  xaxis
yes? DEF AXIS/y=1:180:1  yaxis
yes? DEF AXIS/Z=0:100:25/DEPTH  zaxis
yes? DEF GRID/X=xaxis/Y=yaxis/Z=zaxis uv_grid
yes? DEF AXIS/Z=0:0:1  zaxis1
yes? DEF GRID/LIKE=uv_grid/Z=zaxis1  uv_grid1

yes? FILE/GRID=uv_grid1/BAD=-999/VAR="u,v"  uv.data1
yes? LET/TITLE="My U data"  u1 = u[G=uv_grid]
yes? LET/TITLE="My V data"  v1 = v[G=uv_grid]
```

```
yes? SAVE/FILE=uv1_5z.cdf/KLIMITS=1:5/K=1  u1, v1

yes? CANCEL DATA uv.data1
yes? DEF AXIS/Z=25:25:1  zaxis1
yes? FILE/GRID=uv_grid1/BAD=-999/VAR="u,v"  uv.data2
yes? SAVE/FILE=uv1_5z.cdf/K=2/APPEND  u1,v1
. . .
```

The NetCDF utilities "ncdump" and "ncgen" can also be combined with a text editor to make final refinements to the NetCDF files created by SAVE. (These utilities are not provided with the Ferret distribution; they can be obtained from unidata.ucar.edu.) Here is a simple example that removes all "history" attributes from a NetCDF file using pipes and the Unix "grep" utility:

```
% ncdump old_file.cdf | grep -v history | ncgen -o new_file.cdf
```

## Ch10 Sec3. WRITING A CONVERSION PROGRAM

There are three steps required to convert data to NetCDF if your data is not already readable by Ferret:

1. Create a CDL (the ASCII NetCDF Description Language) file that describes the axes, grids, and variables of the desired output data set. **Note**: Ferret itself often provides the simplest way to create the CDL file (see the following section).

2. Convert this CDL file into a NetCDF file with the ncgen utility.

3. Create a program that will read your particular data and insert them into the NetCDF file. The ncgen utility will create most of the FORTRAN or C code needed for this task.

The file converting_to_netcdf.f which is located in the Ferret documentation directory ($FER_DIR/doc) contains a complete description and example of these three steps. The remainder of this section provides further details.

## Ch10 Sec3.1. Creating a CDL file with Ferret

Suppose that we wish to create a CDL file to describe a data set entitled "My Global Data" which contains variables u and v in cm/sec on a 5×5 degree global lat/long grid. The following commands would achieve the goal with Ferret doing the majority of the work:

- From Ferret issue the commands

```
DEFINE AXIS/X=2.5E:2.5W:5/UNITS=degrees xlong
DEFINE AXIS/Y=87.5S:87.5N:5/UNITS=degrees ylat
DEFINE GRID/X=xlong/Y=ylat my_grid
LET shape_2d =  x[G=my_grid]+y[G=my_grid]
LET U = 1/0*SHAPE_2D
```

```
LET V = 1/0*SHAPE_2D
SET VARIABLE/TITLE="Zonal Velocity"/UNITS="cm/sec" u
SET VARIABLE/TITLE="Meridional Velocity"/UNITS="cm/sec" v
SAVE/FILE=my_file.cdf/TITLE="My Global Data" u,v
QUIT
```

- From Unix issue the command

```
ncdump -c my_file.cdf > my_file.cdl
```

The resulting file my_file.cdl is ready to use or to make final modifications to with an editor.

## Ch10 Sec3.2. The CDL file

A CDL file consists of three sections: Dimensions, Variables, and Data. All of the following text in `Courier font` constitutes a real CDL file; it can be copied verbatim and used to generate a NetCDF file. The full text of this file is included with the Ferret distribution as $FER_DIR/doc/converting_to_netcdf.basic.

```
netcdf converting_to_netcdf.basic {
```

### Ch10 Sec3.2.1. Dimensions

In this section, the sizes of the grid dimensions are specified. One of these dimensions can be of "unlimited" size (i.e., it can grow).

```
Dimensions:

      lon   = 160 ;      //  longitude
      lat   = 100 ;      //  latitude
      depth = 27 ;       //  depth
      time  = unlimited ;
```

These are essentially parameter statements that assign certain numbers that will be used in the Variables section to define axes and variable dimensions. The "//" is the CDL comment syntax.

### Ch10 Sec3.2.2. Variables

Variables, variable attributes, axes, axis attributes, and global attributes are specified.

```
variables:

     float temp(time, depth, lat, lon) ;
           temp: long_name = "TEMPERATURE" ;
           temp: units     = "deg. C" ;
           temp: _FillValue = 1E34 ;
     float salt(time, depth, lat, lon) ;
```

```
salt: long_name = "(SALINITY(ppt) - 35) /1000" ;
salt: units     = "frac. by wt. less .035" ;
salt: _FillValue = -999. ;
```

The declaration "float" indicates that the variable is to be stored as single precision, floating point (32-bit IEEE representation). The declarations "long" (32-bit integer), "short" (16-bit integer), "byte" (8-bit integer) and "double" (64-bit IEEE floating point) are also supported by Ferret. Note that although these data types may result in smaller files, they will not affect Ferret's memory usage, as all variables are converted to "float" internally as they are read by Ferret.

Variable names in NetCDF files should follow the same guidelines as Ferret variable names:

- case insensitive (avoid names that are identical apart from case)
- 1 to 24 characters (letters, digits, $ and _) beginning with a letter
- avoid reserved names (I, J, K, L, X, Y, Z, T, XBOX, ...)

The _FillValue attribute informs Ferret that any data value matching this value is a missing (invalid) data point. For example, an ocean data set may label land locations with a value such as 1E34. By identifying 1E34 as a fill value, Ferret knows to ignore points matching this value. The attribute "missing_value" is similar to "_FillValue" when reading data; but "_FillValue" also specifies a value to be inserted into unspecified regions during file creation. You may specify two distinct flags for invalid data in the same variable by using "_FillValue" and "missing_value" together.

Ferret variables may have from 1 to 4 dimensions. If any of the axes have the special interpretations of: 1) latitude, 2) longitude, 3) depth, or 4) time (date), then the relative order of those axes in the CDL variable declaration must be T, then Z, then Y, and then X, as above. Any of these special axes can be omitted and other axes (for example, an axis called "distance") may be inserted between them.

```
axis definitions:

     double lon(lon) ;
            lon: units = "degrees";
     double lat(lat) ;
            lat: units = "degrees";
     double depth(depth) ;
            depth: units = "meters";
     double time(time) ;
            time: units = "seconds since 1972-01-01";
```

Axes are distinguished from other 1-dimensional NetCDF variables by their variable names being identical to their dimension names. Special axis interpretations are inferred by Ferret through a variety of "clues."

Date/time axes are inferred by units of "years," "days," "hours," "minutes," or "seconds," or by axis names "time," "date," or "t" (case-insensitive). Calendar date formatting requires the "units" attribute to be formatted with both a valid time unit and "since dd-mm-yyyy".

Vertical axes are identified by axis names containing the strings “depth”, “elev”, or “z”, or by units of “millibars.” Depth axes are positive downward by default. The attribute positive= “down” can also be used to create a downward-pointing axis.

Latitude axes are inferred by units of “degrees” or “latitude” with axis names containing the strings “lat” or “y”. Longitude axes are inferred by units of “degrees” or “longitude” with axis names containing the strings “lon” or “x”.

Global attributes, or attributes that apply to the entire data set, can be specified as well.

```
global attributes:
     :title = "NetCDF Example";
     :message =   "This message will be displayed when the CDF file is
                   opened by Ferret";
     :history =   "Documentation on the origins and evolution of this
data
                   set or variable";
```

## Ch10 Sec3.2.3. Data

In this section, values are assigned to grid coordinates and to the variables of the file. Below are 100 latitude coordinates entered (in degrees) into the variable axis “lat”, 160 longitude coordinates in “lon”, and 27 depth coordinates (in meters) in “depth”. Longitude coordinates must be specified with 0 at Greenwich, continuous across the dateline, with positive eastward (modulo 360).

```
Data:

lat=
-28.8360729218,-26.5299491882,-24.2880744934,-22.1501560211,-20.1513576
50,
-18.3207626343,-16.6801033020,-15.2428140640,-14.0134353638,-12.9874248
50,
-12.1513509750,-11.4834814072,-10.9547319412,-10.5299386978,-10.1693935
39,
-9.8333206177,-9.4999876022,-9.1666536331,-8.8333196640,-8.4999856949,
-8.1666526794,-7.8333187103,-7.4999847412,-7.1666512489,-6.8333182335,
-6.4999852180,-6.1666517258,-5.8333182335,-5.4999852180,-5.1666517258,
-4.8333187103,-4.4999852180,-4.1666517258,-3.8333187103,-3.4999852180,
-3.1666517258,-2.8333184719,-2.4999852180,-2.1666519642,-1.8333185911,
-1.4999852180,-1.1666518450,-0.8333183527,-0.4999849498,-0.1666515470,
0.1666818559,0.5000152588,0.8333486915,1.1666821241,1.5000154972,
1.8333489895,2.1666824818,2.5000159740,2.8333494663,3.1666829586,
3.5000162125,3.8333497047,4.1666831970,4.5000162125,4.8333497047,
5.1666831970,5.5000162125,5.8333497047,6.1666827202,6.5000162125,
6.8333497047,7.1666827202,7.5000166893,7.8333501816,8.1666841507,
8.5000181198,8.8333511353,9.1666851044,9.5000190735,9.8333530426,
10.1679363251,10.5137376785,10.8892869949,11.3138961792,11.8060989380,
12.3833675385,13.0618314743,13.8560228348,14.7786512375,15.8403968811,
17.0497493744,18.4128704071,19.9334945679,21.6128730774,23.4497566223,
25.4404067993,27.5786647797,29.8560409546,32.2618522644,34.7833900452,
37.4061241150,40.1139259338,42.8893203735,45.7137718201,48.5679702759;
lon=
130.5,131.5,132.5,133.5,134.5,135.5,136.5,137.5,138.5,139.5,140.5,141.5
,142.5,143.5,144.5,145.5,146.5,147.5,148.5,149.5,150.5,151.5,152.5,153.
```

```
5,154.5,155.5,156.5,157.5,158.5,159.5,160.5,161.5,162.5,163.5,164.5,165
.5,166.5,167.5,168.5,169.5,170.5,171.5,172.5,173.5,174.5,175.5,176.5,17
7.5,178.5,179.5,180.5,181.5,182.5,183.5,184.5,185.5,186.5,187.5,188.5,1
89.5,190.5,191.5,192.5,193.5,194.5,195.5,196.5,197.5,198.5,199.5,200.5,
201.5,202.5,203.5,204.5,205.5,206.5,207.5,208.5,209.5,210.5,211.5,212.5
,213.5,214.5,215.5,216.5,217.5,218.5,219.5,220.5,221.5,222.5,223.5,224.
5,225.5,226.5,227.5,228.5,229.5,230.5,231.5,232.5,233.5,234.5,235.5,236
.5,237.5,238.5,239.5,240.5,241.5,242.5,243.5,244.5,245.5,246.5,247.5,24
8.5,249.5,250.5,251.5,252.5,253.5,254.5,255.5,256.5,257.5,258.5,259.5,2
60.5,261.5,262.5,263.5,264.5,265.5,266.5,267.5,268.5,269.5,270.5,271.5,
272.5,273.5,274.5,275.5,276.5,277.5,278.5,279.5,280.5,281.5,282.5,283.5
,284.5,285.5,286.5,287.5,288.5,289.5;
depth=
5.0,15.0,25.0,35.0,45.0,55.0,65.0,75.0,85.0,95.0,106.25,120.0,136.25,15
5.0,177.5,205.0,240.0,288.5,362.5,483.5,680.0,979.5,1395.5,1916.0,2524.
0,3174.0,3824.0;  }
```

To use this CDL file type:

```
% ncgen -o my_data.cdf converting_to_netcdf.basic
```

This will create a file called “my_data.cdf” to which data can be directed (see next section).

Another NetCDF command, “ncdump”, can be used to generate a CDL file from an existing NetCDF file. The command

```
% ncdump -h my_data.cdf
```

will list the CDL representation of a preexisting my_data.cdf without the Data section, while

```
% ncdump my_data.cdf
```

will list the CDL file with the Data section. The command

```
% ncdump -c my_data.cdf
```

will create a CDL file in which only coordinate variables are included in the Data section. The listed output can be redirected to a file as in the command

```
% ncdump -c my_data.cdf > my_data.cdl
```

## Ch10 Sec3.3. Standardized NetCDF attributes

A document detailing the COARDS NetCDF conventions to which the Ferret program adheres are available on line through the Ferret home page on the World Wide Web, at
http://ferret.wrc.noaa.gov/noaa_coop/ coop_cdf_profile.html and at

http://www.unidata.ucar.edu/packages/netcdf/conventions.html

The following are the attributes most commonly used with Ferret. They are described in greater detail in the reference document named above.

**Global Attributes**

:title = “my data set title”
:history = “general background information”

**Data Variable Attributes**

long_name = “title of my variable”
units = “units for this variable”
_FillValue = missing value flag
missing_value = alternative missing value flag
scale_factor = (optional) the data are to be multiplied by this factor
add_offset = (optional) this number is to be added to the data

**Special Coordinate Variable Attributes**

time_axis:units = “seconds since 1992-10-8 15:15:42.5 -6:00"; // example
y_axis:units = “degrees_north”
x_axis:units = “degrees_east”
z_axis:positive = “down”; // to indicate preferred plotting orientation
my_axis:point_spacing = “even”; // improved performance if present

## Ch10 Sec3.4. Directing data to a CDF file

The following is an example program which can be used on-line to convert existing data sets into NetCDF files. It also should provide guidance on sending data generated by numerical models directly to NetCDF files. This program assumes you have created the NetCDF file as described in the previous section. It is included in the distribution as $FER_DIR/doc/converting_to_netcdf.f.

```
   program converting_to_netcdf

c written by Dan Trueman
c updated 4/94 *sh*

c This program provides a model for converting a data set to NetCDF.
c The basic strategy used in this program is to open an existing NetCDF
c file, query the file for the ID’s of the variables it contains, and
c then write the data to those variables.

c The output NetCDF file must be created **before** this program is run.
c The simplest way to do this is to cd to your scratch directory and
c    % cp $FER_DIR/doc/converting_to_netcdf.basic converting_to_netcdf.cdl
c and then edit converting_to_netcdf.cdl (an ASCII file) to describe YOUR
c data set. If your data set requires unequally spaced axes,
climatological c time axes, staggered grids, etc. then
converting_to_netcdf.supplement may c be a better starting point then the
“basic” file used above.
c After you edit converting_to_netcdf.cdl then create the NetCDF file with
c the command
```

```
c    % ncgen -o converting_to_netcdf.cdf converting_to_netcdf.cdl

c Now we will read in **your** data (gridded oceanic temperature and
c salt in this example) and write it out into the NetCDF file
c converting_to_netcdf.cdf.  Note that the axis coordinates can be written
c out exactly the same methodology - including time step values (as
below).
**********************************************************************
c An alternative to modifying this program is to use the command:

c     ncgen -f converting_to_netcdf.cdl

c This will create a large source code to which select lines can
c be added to write out your data.
**********************************************************************
c To compile and link converting_to_netcdf.f, use:

c     f77 -o converting_to_netcdf converting_to_netcdf.f -lnetcdf

**********************************************************************
c include file necessary for NetCDF

   include 'netcdf.inc'    ! may be found in $FER_DIR/fmt/cmn
**********************************************************************
c parameters relevant to the data being read in
c THESE NORMALLY MATCH THE DIMENSIONS IN THE CDL FILE
c (except nt which may be "unlimited")

   integer imt, jmt, km, nt, lnew, inlun
   parameter (imt=160, jmt=100, km=27, nt=5)

c imt is longitude, jmt latitude, km depth, and nt number of time steps
**********************************************************************
c variable declaration

   real temp(imt,jmt,km),salt(imt,jmt,km),time_step

   integer cdfid, rcode
c           ** cdfid = id number for the NetCDF file my_data.cdf
c     ** rcode = error id number

   integer tid, sid, timeaxid
c           ** tid = variable id number for temperature
c           ** sid = variable id number for salt
c           ** timeaxid = variable id for the time axis
   integer itime
c           ** itime = index for do loop
**********************************************************************
c dimension corner and step for defining size of gridded data

   integer corner(4)
   integer step(4)

c corner and step are used to define the size of the gridded data
c to be written out.  Since temp and salt are four dimensional arrays,
c corner and step must be four dimensions as well.  In each output
c to my_data.cdf within the do loop, the entire array of data (160 long.
c pts, 100 lat. pts., 27 depth pts.) will be written for one time step.
c Corner tells NetCDF where to start, and step indicates how many steps
c in each dimension to take.

        data corner/1, 1, 1, -1/      ! -1 is arbitrary; the time value

! of corner will be initialized
```

```
                                        ! within the do loop.

        data step/imt, jmt, km, 1/    ! NOT /1, km, jmt, imt/

c ***NOTE*** Since Fortran and C access data differently, the order of
c the variables in the Fortran code must be opposite that in the CDL
c file.  In Fortran, the first index varies fastest while in C, the
c last index varies fastest.
*************************************************************************
c initialize cdfid by using ncopn

   cdfid = ncopn('converting_to_netcdf.cdf', ncwrite, rcode)
   if (rcode.ne.ncnoerr) stop 'error with ncopn'

*************************************************************************
c get variable id's by using ncvid
c THE VARIABLE NAMES MUST MATCH THE CDL FILE (case sensitive)

   tid = ncvid(cdfid, 'temp', rcode)
   if (rcode.ne.ncnoerr) stop 'error with tid'
   sid = ncvid(cdfid, 'salt', rcode)
   if (rcode.ne.ncnoerr) stop 'error with sid'
   timeaxid = ncvid(cdfid, 'time', rcode)
   if (rcode.ne.ncnoerr) stop 'error with timeaxid'
*************************************************************************
c this is a good place to open your input data file
   ! OPEN (FILE=my_data.dat,STATUS='OLD')
*************************************************************************
c begin do loop.  Each step will read in one time step of data
c and then write it out to my_data.cdf.


   do 10 itime = 1, nt

   corner(4) = itime             ! initialize time step in corner
   time_step = float(itime)      ! or you may read this from your file

* insert your data reading routine here
!     CALL READ_MY_DATA(temp,salt)  ! you write this

   write (6,*) 'writing time step: ',itime, time_step  ! diagnostic output

   call ncvpt(cdfid,tid,corner,step,temp(1,1,1),rcode) ! write data to
   if (rcode.ne.ncnoerr) stop 'error with t-put'
   call ncvpt(cdfid,sid,corner,step,salt(1,1,1),rcode) ! my_data.cdf with
   if (rcode.ne.ncnoerr) stop 'error with s-put'
   call ncvpt1(cdfid,timeaxid,itime,time_step,rcode)   ! ncvpt
   if (rcode.ne.ncnoerr) stop 'error with timax-put'

c ncvpt1 writes a single data point to the specified location within
c timeaxid. The itime argument in ncvpt1 specifies the location within
c time to write.
c float(itime) is used (rather than simply itime) so the type matches the
c type of time declared in the CDL file.

10    continue
*************************************************************************
c close my_data.cdf using ncclos
   call ncclos(cdfid, rcode)
   if (rcode.ne.ncnoerr) stop 'error with ncclos'
*************************************************************************
   stop
   end
```

## Ch10 Sec3.5. Advanced NetCDF procedures

This section describes:

1. Setting up a CDL file capable of handling data on staggered grids.
2. Defining coordinate systems such that the data in the NetCDF file may be regarded as hyperslabs of larger coordinate spaces.
3. Defining boundaries between unevenly spaced axis coordinates (used in numerical integrations).
4. Setting up "modulo" axes such as climatological time and longitude.
5. Converting dates into numerical data appropriate for a NetCDF time axis.

The final section of this chapter contains the text of the CDL file for the example we use throughout this section.

In this sample data set, we will consider wind, salt, and velocity calculated using a staggered-grid, finite-difference technique. The wind data is limited to the surface layer of the ocean (i.e., normal to the depth axis). We will also consider the salt data to be limited to a narrow slab from 139E to 90W (I=10 to 140), 32.5N to 34.9N (J=80 to 82), and for all depth and time values.

### Ch10 Sec3.5.1. Staggered grid

Ferret permits each variable of a NetCDF file to be defined on distinct axes and grids. Staggered grids are a straightforward application of this principle. Dimensions for each grid axis must be defined, the axes themselves must be defined (in Variables), and the coordinate values for each axis must be initialized (in Data). In the case of the example we use throughout this and the next section, there are two grids—a wind grid, and a velocity grid; slon, slat and sdepth are defined for the wind grid, and ulon, ulat, and wdepth for the velocity grid. The variables are then given dimensions to place them in their proper grids (i.e., wind(time, sdepth, slat, slon)).

### Ch10 Sec3.5.2. Hyperslabs

There are a number of steps required to set up a NetCDF data set that represents a hyperslab of data from a larger grid definition (a parent grid).

1. Define a dimension named "grid_definition." This dimension should be set equal to 1.

2. Define parent grids in Variables with the argument "grid_definition".

```
char wind_grid(grid_definition) ;
char salt_grid(grid_definition) ;
```

3. Define the 4 axes of the parent grids using the “axes” attribute.

```
wind_grid: axes = "slon slat normal time" ;
salt_grid: axes = "slon slat sdepth time" ;
```

The arguments are always a list of four axis names. Note that the order of arguments is opposite that in the variable declaration. The argument “normal” indicates that wind_grid is normal to the depth axis.

4. Define the variables that are hyperslabs of these grids with the proper dimensions.

```
float wind(time, slat, slon) ;
float salt(time, sdepth, slat80_82, slon10_140) ;
```

where the dimension slat80_82 = 3 and slon10_140 = 131. Optionally, these axes may be defined themselves with the attribute “child_axis”.

```
float slat80_82(slat80_82) ;
slat80_82: child_axis = " " ;
```

These “child axes” need not be initialized in data, nor do edges need to be defined for them; Ferret will retrieve this information from the parent axis definitions. However, it is recommended that they be initialized to accommodate other software that may not recognize parent grids.

5. Use the “parent_grid” variable attribute to point to the parent grid.

```
wind: parent_grid = "wind_grid"
salt: parent_grid = "salt_grid"
```

6. Also, as a variable attribute, define the index range of interest within the parent grid.

```
wind: slab_min_index = 1s, 1s, 1s, 0s ;
wind: slab_max_index = 160s, 100s, 1s, 0s ;
salt: slab_min_index = 10s, 80s, 1s, 0s ;
salt: slab_max_index = 140s, 82s, 27s, 0s ;
```

The “s” after each integer indicates a “short” 16-bit integer rather than the default “long” 32-bit integer. If an axis dimension is designated as “unlimited” then the index bounds for this axis must be designated as “0s”.

These attributes will effectively locate the wind and salt data within the parent grid.

## Ch10 Sec3.5.3. Unevenly spaced coordinates

For coordinate axes with uneven spacing, the boundaries between each coordinate can be indicated by pointing to an additional axis that contains the locations of the boundaries. The dimension of this “edge” axis is necessarily one larger than the coordinate axis concerned. If edges

are not explicitly defined for an unevenly spaced axis, the midpoint between coordinates is assumed by default.

1. Define a dimension one larger than the coordinate axis. For the sdepth axis, with 27 coordinates, define:

```
sdepth_edges = 28 ;
```

2. Define an axis called sdepth_edges.
3. Initialize this axis with the desired boundaries (in Data).
4. As an attribute of the main axis, point to edges list:

```
sdepth: edges = "sdepth_edges" ;
```

## Ch10 Sec3.5.4. Evenly spaced coordinates (long axes)

If the coordinate axes are evenly spaced, the attribute "point spacing" should be used:

```
slat: point_spacing = "even" ;
```

When used, this attribute will improve memory use efficiency in Ferret. This is especially important for very large axes—10,000 points or more.

## Ch10 Sec3.5.5. "Modulo" axes

The "modulo" axis attribute indicates that the axis wraps around, the first point immediately following the last. The most common uses of modulo axes are:

1. longitude axes for globe-encircling data
2. time axes for climatological data

```
time: modulo = " " ;  // any arbitrary string is allowed
```

If the climatological data occurs in the years 0000 or 0001 then the year will be omitted from Ferret's output format.

## Ch10 Sec3.5.6. Reversed-coordinate axes

NetCDF axes may contain monotonically decreasing axis coordinates instead of monotonically increasing coordinates. Ferret will hide this aspect of the file data ordering.

## Ch10 Sec3.5.7. Converting time word data to numerical data

To set up a time axis for data represented as dates (e.g., “1972 January 15 at 12:15”) it is necessary to determine a numerical representation for each of the dates. Ferret can assist with this process, as the following example shows.

Suppose the data are 6-hourly observations from 1-JAN-1991 at 12:00 to 15-MAR-1991 at 18:00. These commands will assist in creating the necessary time axis for a NetCDF file:

```
yes? DEFINE
AXIS/T="1-JAN-1991:12:00":"15-MAR-1991:18:00":6/UNITS=hours\
      my_time
yes? DEFINE GRID/T=my_time tgrid
yes? SET REGION/T="1-JAN-1991:12:00":"15-MAR-1991:18:00"
yes? LIST T[g=tgrid]                                !to see the time
values
yes? SAVE/FILE=my_time.cdf T[g=tgrid]
```

The file my_time.cdf now contains a model of exactly the desired time axis. Use the Unix command

```
% ncdump my_time.cdf > my_time.cdl
```

to obtain the time axis definition as text that can be inserted into your CDL file.

## Ch10 Sec3.6. Example CDL file

The following is an example CDL file utilizing many of the features described in the preceding section.

```
netcdf converting_to_netcdf_supplement {
//      CONVERTING DATA TO THE “NETWORK COMMON DATA FORM” (NetCDF):
//                          A SUPPLEMENT
//
// NOAA PMEL Thermal Modeling and Analysis Project (TMAP)
// Dan Trueman, Steve Hankin
// last revised: 1 Dec 1993:  slat80_82 and slon10_140 coordinates included
//
// I. INTRODUCTION
//
// This supplement to “Converting Data to the Network Common Data Form:
// an Introduction” describes:
//
//      1. How to set up a cdl file capable of handling data
//         on staggered grids.
//      2. How to define coordinate systems such that the data
//         in the NetCDF file may be regarded as hyperslabs of
//         larger coordinate spaces.
//      3. How to define variables of 1, 2, or 3 dimensions.
//      4. How to define boundaries between unevenly spaced axis
//         coordinates (used in numerical integrations).
//      5. How to set up climatological “modulo” time axes.
//      6. How to convert time word data into numerical data
//         appropriate for NetCDF.
//
// In this sample data set, we will consider wind, salt, and
// velocity calculated using a staggered-grid, finite-difference
```

```
// technique. The wind data is naturally limited to the surface
// layer of the ocean (i.e. normal to the  depth axis).  We will
// also consider the salt data to be limited to a narrow slab from
// 139E to 90W (I=10 to 140), 32.5N to 34.9N (J=80 to 82), and for
// all depth and time values.
//
// II. STAGGERED GRIDS
//
// Dealing with staggered grids is fairly straightforward.  Dimensions
// for each grid axis must be defined, the axes themselves must be
// defined (in Variables), and the coordinate values for each axis must
// be initialized (in Data).  In this case, there are two grids, a
// wind grid, and a velocity grid, so tlon, tlat and tdepth are
// defined for the wind grid, and ulon, ulat, and udepth for the velocity
// grid.  The variables are then given arguments to place them in their
// proper grids (i.e. wind(time, sdepth, slat, slon)).
//
// III. HYPERSLABS
//
// There are a number of steps required to set up a NetCDF data set that
// represents a hyperslab of data from a larger grid definition.
//
//      1. Define a dimension named "grid_definition".  This dimension
//         should be set equal to 1.
//      2. Define parent grids in Variables with the argument
//         "grid_definition".
//
//              char wind_grid(grid_definition) ;
//              char salt_grid(grid_definition) ;
//
//      3. Define the 4 axes of the parent grids using the "axes" attribute.
//
//                  wind_grid: axes = "slon slat normal time" ;
//                  salt_grid: axes = "slon slat sdepth time" ;
//
//          Note that the order of arguments is opposite that in the
//          variable declaration.  The argument "normal" indicates that
//          wind_grid is normal to the depth axis.
//
//       4. Define the variables which are hyperslabs of these grids with
//          the proper dimensions.
//
//              float wind(time, slat, slon) ;
//              float salt(time, sdepth, slat80_82, slon10_140) ;
//
//         where slat80_82 = 3 and slon10_140 = 131. The axis names are
///         arbitrary - chosen for readability. These axes (child axes)
//          must be defined with the attribute "child_axis" as follows:
//
//              float slat80_82(slat80_82) ;
//                    slat80_82: child_axis = " " ;
//
//         These "child axes" need not be initialized in Data, nor do their

//         edges need be defined; Ferret retrieves this information from
//         the parent axes.
//
//      5. Use the "parent_grid" variable attribute to point to the
//         parent grid.
//
//                  wind: parent_grid = "wind_grid"
//
//      6. Also as a variable attribute, define the index range of interest
//         within the parent grid.
//
//                  wind: slab_min_index = 1s, 1s, 1s, 0s ;
//                  wind: slab_max_index = 160s, 100s, 1s, 0s ;
//                  salt: slab_min_index = 10s, 80s, 1s, 0s ;
//                  salt: slab_max_index = 140s, 82s, 27s, 0s ;
//
//         The "s" after each integer indicates a "short" 16-bit integer
```

```
//         rather than the default "long" 32-bit integer.  If an axis
//         dimension is designated as "unlimited" then the index bounds
//         for this axis must be designated as "0s".
//
// These commands will effectively locate the wind and salt data within
// the full grid.
//
// IV. VARIABLES OF 1, 2, or 3 DIMENSIONS
//
// One, two, or three dimensional variables may be set up in one of
// two ways - either using the parent_grid and child_axis attributes
// as illustrated in the 3-dimensional variable "wind" from the hyperslab
// example, above, or by selecting axis names and units that provide
// Ferret with adequate hints to map this variable onto 4-dimensional
// space and time.  The following hints are recognized by Ferret:
//
//      Units of days, hours, minutes, etc. or an axis name of "TIME", "DATE"
//         implies a time axis.
//      Units of "degrees xxxx" where "xxxx" contains "lat" or "lon" implies
//         a latitude or longitude axis, respectively.
//      Units of "degrees" together with an axis name containing "LAT" or
//         "Y" implies a latitude axis else longitude is assumed.
//      Units of millibars, "layer" or "level" or an axis name containing
//         "Z" or "ELEV" implies a vertical axis.
//
// V. UNEVENLY SPACED COORDINATE BOUNDARIES
//
// For coordinate axes with uneven spacing, the boundaries between each
// coordinate can be indicated by pointing to an additional axis that
// contains the locations of the boundaries.  The dimension of this "edge"
// axis will necessarily be one larger than the coordinate axis concerned.
// If edges are not defined for an unevenly spaced axis, the midpoint
// between coordinates will be assumed by default.
//
//      1. Define a dimension one larger than the coordinate axis.  For
//         the sdepth axis, with 27 coordinates, define:
//
//              sdepth_edges = 28 ;
//
//      2. Define an axis called sdepth_edges.
//      3. Initialize this axis appropriately (in Data).
//      4. As a sdepth axis attribute, point to sdepth_edges:
//
//                  sdepth: edges = "sdepth_edges" ;
//
// If the coordinate axes are evenly spaced, the attribute "point spacing"
// should be used:
//
//                  slat: point_spacing = "even" ;
//
// When used, this attribute will improve memory use efficiency in Ferret.
//
// VI. CLIMATOLOGICAL "MODULO" AXES
//
// The "modulo" axis attribute indicates that the axis wraps around,
// the first point immediately following the last.  The most common
// uses of modulo axes are:
//
//      1. As longitude axes for globe-encircling data.
//      2. As time axes for climatological data.
//
//                  time: modulo = " " ; // any arbitrary string is allowed
//
// If the climatological data occurs in the years 0000 or 0001 then Ferret
// will omit the year from the output formatting.
//
// VII. CONVERTING TIME WORD DATA TO NUMERICAL DATA
//
// If the time data being converted to NetCDF format exists in string format
// (i.e. 1972 - JANUARY 15 2:15:00), rather than numerical format (i.e. 55123
// seconds) a number of TMAP routines are available to aid in the conversion
// process.  The steps required for conversion are as follows:
```

```
//
//      1. Break the time string into its 6 pieces.  If the data is of the
//         form dd-mmm-yyyy:hh:mm:dd, the TMAP routine "tm_break_date.f" can
//         be used.
//      2. Choose a time_origin before the beginning of the time data to
//         assure that all time values are positive.  i.e. if the data begins
//         at 15-JAN-1982:05:30:00, choose a time origin of
//         15-JAN-1981:00:00:00. This time_origin should then be an attribute
//         of the time axis variable in the CDL file.
//      3. Produce numerical time data by using "tm_sec_from_bc.f", which
//         calculates the number of seconds between 01-01-0000:00:00:00 and
//         the date specified.  Continuing the example from (2), the time value
//         for the first time step with respect to the time_origin could be
//         calculated as follows:
//
//         time(1) = tm_sec_from_bc(1982, 1, 15, 5, 30, 0) -
//                   tm_sec_from_bc(1981, 1, 15, 0, 0, 0)
//
//         or more generally
//
//         time(n)=tm_sec_from_bc(nyear,nmonth,nday,nhour,nminute,nsecond) -
//                 tm-sec_from_bc(oyear,omonth,oday,ohour,ominute,osecond)
//
//         where nyear is the year for the nth time step and oyear is the year
//         of time_origin.
//
// VII. EXAMPLE CDL FILE dimensions:

// staggered grid dimension definitions:

             slon = 160 ;    // wind/salt longitude dimension
             ulon = 160 ;    // velocity longitude dimension
             slat = 100 ;    // wind/salt latitude dimension
             ulat = 100 ;    // velocity latitude dimension
             sdepth = 27 ;   // salt depth dimension
             wdepth = 27 ;   // velocity depth dimension

             slon10_140 = 131 ;      // for salt hyperslab
             slat80_82 = 3 ;         // for salt hyperslab
             time = unlimited ;

 // grid_definition is the dimension name to be used for all grid definitions

             grid_definition = 1 ;

// edge dimension definitions:

             sdepth_edges = 28 ;
             wdepth_edges = 28 ;

variables:

  // variable definitions:

             float wind(time, slat, slon) ;  // 3-dimensional variable
                          wind: parent_grid = "wind_grid" ;
                          wind: slab_min_index = 1s, 1s, 1s, 0s ;
                          wind: slab_max_index = 160s, 100s, 1s, 0s ;
                          wind: long_name = "WIND" ;
                          wind: units     = "deg. C" ;
                          wind: _FillValue = 1E34f ;
             float salt(time, sdepth, slat80_82, slon10_140) ; // 4-dim.
Variable
                          salt: parent_grid = "salt_grid" ;
                          salt: slab_min_index = 10s, 80s, 1s, 0s ;
                          salt: slab_max_index = 140s, 82s, 27s, 0s ;
                          salt: long_name = "(SALINITY(ppt) - 35) /1000" ;
                          salt: units     = "frac. by wt. less .035" ;
                          salt: _FillValue = -999.f ;

             float u(time, sdepth, ulat, ulon) ;
```

```
                        u: long_name    = "ZONAL VELOCITY" ;
                        u: units        = "cm/sec" ;
                        u: _FillValue = 1E34f ;
                float v(time, sdepth, ulat, ulon) ;
                        v: long_name    = "MERIDIONAL VELOCITY" ;
                        v: units        = "cm/sec" ;
                        v: _FillValue = 1E34f ;
                float w(time, wdepth, slat, slon) ;
                        w: long_name    = "VERTICAL VELOCITY" ;
                        w: units        = "cm/sec" ;
                        w: _FillValue = 1E34f ;

    // axis definitions:

                float slon(slon) ;
                        slon: units = "degrees" ;
                        slon: point_spacing = "even" ;
                float ulon(ulon) ;
                        ulon: units = "degrees" ;
                        ulon: point_spacing = "even" ;
                float slat(slat) ;
                        slat: units = "degrees" ;
                        slat: point_spacing = "even" ;
                float ulat(ulat) ;
                        ulat: units = "degrees" ;
                      ulat: point_spacing = "even" ;
                float sdepth(sdepth) ;
                        sdepth: units = "meters" ;
                        sdepth: positive = "down" ;
                        sdepth: edges = "sdepth_edges" ;
                float wdepth(wdepth) ;
                        wdepth: units = "meters" ;
                        wdepth: positive = "down" ;
                        wdepth: edges = "wdepth_edges" ;
                float time(time) ;
                        time: modulo = " " ;
                      time: time_origin = "15-JAN-1981:00:00:00" ;
                        time: units = "seconds" ;

    // child grid definitions:

                float slon10_140(slon10_140) ;
                        slon10_140: child_axis = " " ;

                        slon10_140: units = "degrees" ;
                float slat80_82(slat80_82) ;
                        slat80_82: child_axis = " " ;
                        slat80_82: units = "degrees" ;

    // edge axis definitions:

                float sdepth_edges(sdepth_edges) ;
                float wdepth_edges(wdepth_edges) ;

    // parent grid definition:

                char wind_grid(grid_definition) ;
                      wind_grid: axes = "slon slat normal time" ;
                char salt_grid(grid_definition) ;
                      salt_grid: axes = "slon slat sdepth time" ;

    // global attributes:
                :title = "NetCDF Title" ;

data:

//  //  ignore this block  //
//This next data entry, for time, should be ignored. Time is initialized here
// only so that Ferret can read test.cdf (the file created by this cdl file)
// with no additional data inserted into it.
Time=1000;
// // end of ignored block //
```

```
slat=
-28.8360729218,-26.5299491882,-24.2880744934,-22.1501560211,-20.1513576508,
-18.3207626343,-16.6801033020,-15.2428140640,-14.0134353638,-12.9874248505,
-12.1513509750,-11.4834814072,-10.9547319412,-10.5299386978,-10.1693935394,
-9.8333206177,-9.4999876022,-9.1666536331,-8.8333196640,-8.4999856949,
-8.1666526794,-7.8333187103,-7.4999847412,-7.1666512489,-6.8333182335,
-6.4999852180,-6.1666517258,-5.8333182335,-5.4999852180,-5.1666517258,
-4.8333187103,-4.4999852180,-4.1666517258,-3.8333187103,-3.4999852180,
-3.1666517258,-2.8333184719,-2.4999852180,-2.1666519642,-1.8333185911,
-1.4999852180,-1.1666518450,-0.8333183527,-0.4999849498,-0.1666515470,
0.1666818559,0.5000152588,0.8333486915,1.1666821241,1.5000154972,
1.8333489895,2.1666824818,2.5000159740,2.8333494663,3.1666829586,
3.5000162125,3.8333497047,4.1666831970,4.5000162125,4.8333497047,
5.1666831970,5.5000162125,5.8333497047,6.1666827202,6.5000162125,
6.8333497047,7.1666827202,7.5000166893,7.8333501816,8.1666841507,
8.5000181198,8.8333511353,9.1666851044,9.5000190735,9.8333530426,
10.1679363251,10.5137376785,10.8892869949,11.3138961792,11.8060989380,
12.3833675385,13.0618314743,13.8560228348,14.7786512375,15.8403968811,
17.0497493744,18.4128704071,19.9334945679,21.6128730774,23.4497566223,
25.4404067993,27.5786647797,29.8560409546,32.2618522644,34.7833900452,
37.4061241150,40.1139259338,42.8893203735,45.7137718201,48.5679702759;
ulat=
-27.6721439362,-25.3877544403,-23.1883945465,-21.1119174957,-19.1907978058,
-17.4507274628,-15.9094810486,-14.5761461258,-13.4507236481,-12.5241250992,
-11.7785758972,-11.1883859634,-10.7210769653,-10.3387994766,-9.9999876022,
-9.6666545868,-9.3333206177,-8.9999866486,-8.6666526794,-8.3333196640,
-7.9999856949,-7.6666517258,-7.3333182335,-6.9999847412,-6.6666512489,
-6.3333182335,-5.9999847412,-5.6666517258,-5.3333182335,-4.9999847412,
-4.6666517258,-4.3333182335,-3.9999849796,-3.6666517258,-3.3333184719,
-2.9999852180,-2.6666519642,-2.3333184719,-1.9999853373,-1.6666518450,
-1.3333184719,-0.9999850392,-0.6666516662,-0.3333182633,0.0000151545,
0.3333485723,0.6666819453,1.0000153780,1.3333487511,1.6666821241,
2.0000154972,2.3333489895,2.6666827202,3.0000162125,3.3333497047,
3.6666829586,4.0000162125,4.3333497047,4.6666827202,5.0000162125,
5.3333492279,5.6666827202,6.0000162125,6.3333492279,6.6666827202,
7.0000157356,7.3333497047,7.6666831970,8.0000171661,8.3333511353,
8.6666841507,9.0000181198,9.3333520889,9.6666860580,10.0000190735,
10.3358526230,10.6916217804,11.0869522095,11.5408391953,12.0713586807,
12.6953773499,13.4282865524,14.2837600708,15.2735414505,16.4072513580,
17.6922454834,19.1334934235,20.7334957123,22.4922523499,24.4072608948,
26.4735546112,28.6837768555,31.0283031464,33.4953994751,36.0713844299,
38.7408676147,41.4869842529,44.2916526794,47.1358833313,50.0000534058;
slon=
130.5,131.5,132.5,133.5,134.5,135.5,136.5,137.5,138.5,139.5,140.5,141.5,
142.5,143.5,144.5,145.5,146.5,147.5,148.5,149.5,150.5,151.5,152.5,153.5,
154.5,155.5,156.5,157.5,158.5,159.5,160.5,161.5,162.5,163.5,164.5,165.5,
166.5,167.5,168.5,169.5,170.5,171.5,172.5,173.5,174.5,175.5,176.5,177.5,
178.5,179.5,180.5,181.5,182.5,183.5,184.5,185.5,186.5,187.5,188.5,189.5,
190.5,191.5,192.5,193.5,194.5,195.5,196.5,197.5,198.5,199.5,200.5,201.5,
202.5,203.5,204.5,205.5,206.5,207.5,208.5,209.5,210.5,211.5,212.5,213.5,
214.5,215.5,216.5,217.5,218.5,219.5,220.5,221.5,222.5,223.5,224.5,225.5,
226.5,227.5,228.5,229.5,230.5,231.5,232.5,233.5,234.5,235.5,236.5,237.5,
238.5,239.5,240.5,241.5,242.5,243.5,244.5,245.5,246.5,247.5,248.5,249.5,
250.5,251.5,252.5,253.5,254.5,255.5,256.5,257.5,258.5,259.5,260.5,261.5,
262.5,263.5,264.5,265.5,266.5,267.5,268.5,269.5,270.5,271.5,272.5,273.5,
274.5,275.5,276.5,277.5,278.5,279.5,280.5,281.5,282.5,283.5,284.5,285.5,
286.5,287.5,288.5,289.5;
ulon=
131.0,132.0,133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,
143.0,144.0,145.0,146.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,
155.0,156.0,157.0,158.0,159.0,160.0,161.0,162.0,163.0,164.0,165.0,166.0,
167.0,168.0,169.0,170.0,171.0,172.0,173.0,174.0,175.0,176.0,177.0,178.0,
179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.0,189.0,190.0,
191.0,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.0,202.0,
203.0,204.0,205.0,206.0,207.0,208.0,209.0,210.0,211.0,212.0,213.0,214.0,
215.0,216.0,217.0,218.0,219.0,220.0,221.0,222.0,223.0,224.0,225.0,226.0,
227.0,228.0,229.0,230.0,231.0,232.0,233.0,234.0,235.0,236.0,237.0,238.0,
239.0,240.0,241.0,242.0,243.0,244.0,245.0,246.0,247.0,248.0,249.0,250.0,
251.0,252.0,253.0,254.0,255.0,256.0,257.0,258.0,259.0,260.0,261.0,262.0,
263.0,264.0,265.0,266.0,267.0,268.0,269.0,270.0,271.0,272.0,273.0,274.0,
```

```
275.0,276.0,277.0,278.0,279.0,280.0,281.0,282.0,283.0,284.0,285.0,286.0,
287.0,288.0,289.0,290.0;
sdepth=
5.0,15.0,25.0,35.0,45.0,55.0,65.0,75.0,85.0,95.0,106.25,120.0,136.25,155.0,
177.5,205.0,240.0,288.5,362.5,483.5,680.0,979.5,1395.5,1916.0,2524.0,3174.0,
3824.0;
sdepth_edges=
0.0,10.0,20.0,30.0,40.0,50.0,60.0,70.0,80.0,90.0,100.0,112.5,127.5,
145.0,165.0,190.0,220.0,260.0,317.0,408.0,559.0,801.0,1158.0,1633.0,2199.0,
2849.0,3499.0,4149.0;
wdepth=
10.0,20.0,30.0,40.0,50.0,60.0,70.0,80.0,90.0,100.0,112.5,127.5,145.0,165.0,
190.0,220.0,260.0,317.0,408.0,559.0,801.0,1158.0,1633.0,2199.0,2849.0,3499.0,
4149.0;
wdepth_edges=
5.0,15.0,25.0,35.0,45.0,55.0,65.0,75.0,85.0,94.375,105.625,119.375,135.625,
153.75,176.25,202.5,235.75,280.0,347.5,460.75,651.25,950.0,1372.75,1895.0,
2524.0,3174.0,3986.5,4311.0;
slon10_140=
      139.5, 140.5, 141.5, 142.5, 143.5, 144.5, 145.5, 146.5, 147.5,
      148.5, 149.5, 150.5, 151.5, 152.5, 153.5, 154.5, 155.5, 156.5, 157.5,
      158.5, 159.5, 160.5, 161.5, 162.5, 163.5, 164.5, 165.5, 166.5, 167.5,
      168.5, 169.5, 170.5, 171.5, 172.5, 173.5, 174.5, 175.5, 176.5, 177.5,
      178.5, 179.5, 180.5, 181.5, 182.5, 183.5, 184.5, 185.5, 186.5, 187.5,
      188.5, 189.5, 190.5, 191.5, 192.5, 193.5, 194.5, 195.5, 196.5, 197.5,
      198.5, 199.5, 200.5, 201.5, 202.5, 203.5, 204.5, 205.5, 206.5, 207.5,
      208.5, 209.5, 210.5, 211.5, 212.5, 213.5, 214.5, 215.5, 216.5, 217.5,
      218.5, 219.5, 220.5, 221.5, 222.5, 223.5, 224.5, 225.5, 226.5, 227.5,
      228.5, 229.5, 230.5, 231.5, 232.5, 233.5, 234.5, 235.5, 236.5, 237.5,
      238.5, 239.5, 240.5, 241.5, 242.5, 243.5, 244.5, 245.5, 246.5, 247.5,
      248.5, 249.5, 250.5, 251.5, 252.5, 253.5, 254.5, 255.5, 256.5, 257.5,
      258.5, 259.5, 260.5, 261.5, 262.5, 263.5, 264.5, 265.5, 266.5, 267.5,
      268.5, 269.5 ;
slat80_82=
      11.8060989379883, 12.3833675384522, 13.0618314743042 ;

}
```

## Ch10 Sec4. CREATING A MULTI-FILE NETCDF DATA SET

Ferret supports collections of NetCDF files that are regarded as a single NetCDF data set. Such data sets are referred to as “MC” (multi CDF) data sets. A descriptor file, in the style of TMAP-formatted data sets. These are FORTRAN NAMELIST-formatted files. Slight variations in syntax exist between systems. The requirements for an MC data set are described in the chapter “Data Set Basics”, section “Multi-file NetCDF data sets”.

A typical MC descriptor file is given below. This file ties into a single data set the 23 files named mtaa063-nc.001 through mtaa063-nc.024. The time steps are encoded in the descriptor file through the S_START and S_END values. Ferret performs sanity checking on the data set by comparing these time coordinates with those found in the data files as the data are read.

```
*****************************************************************
****
*        NOAA/PMEL Tropical Modeling and Analysis Program, Seattle, WA.
*
*                        created by MAKE_DESCRIPT rev. 4.01
*
*****************************************************************
****
 $FORMAT_RECORD
```

```
     D_TYPE               = '  MC',
     D_FORMAT             = '  1A',
     D_SOURCE_CLASS       = 'MODEL OUTPUT',
 $END
 $BACKGROUND_RECORD
     D_EXPNUM             = '0063',
     D_MODNUM             = '  AA',
     D_TITLE              = 'MOM model output forced by Sadler winds',
     D_T0TIME             = '14-JAN-1980 14:00:00',
     D_TIME_UNIT          = 3600.0,
     D_TIME_MODULO        = .FALSE.,
     D_ADD_PARM           = 15*' ',
 $END
 $MESSAGE_RECORD
     D_MESSAGE            = ' ',
     D_ALERT_ON_OPEN      = F,
     D_ALERT_ON_OUTPUT    = F,
 $END
*************************************************
 $EXTRA_RECORD
 $END

 $STEPFILE_RECORD
     s_filename           = 'mtaa063-nc.001',
     S_AUX_SET_NUM        = 0,
     S_START              = 17592.0,
     S_END                = 34309.0,
     S_DELTA              = 73.0,
     S_NUM_OF_FILES       = 23,
     S_REGVARFLAG         = ' ',
 $END
 ************************************************
 $STEPFILE_RECORD
     s_filename           = '**END OF STEPFILES**'

 $END
 ************************************************
```

# Chapter 11: WRITING EXTERNAL FUNCTIONS

## Ch11 Sec1. OVERVIEW

External functions are user-written Fortran routines which are called from the Ferret command line just as internal Ferret functions (e.g. SIN, COS) are invoked. For example, you might create a routine to compute the amplitudes of the Fourier transform of a time series (the periodogram) and name your function "FFT_AMP". In Ferret you would use it like this:

```
LET my_fft = FFT_AMP(my_time_series)
```

Once the variable my_fft is defined, it can be used in other expressions, plotted, etc. External functions can be used in every way that Ferret internal functions are used and are distinguished only by their appearance after internal functions when the user issues a SHOW FUNC command.

A Ferret external function uses input arguments defined in a Ferret session and computes a result with user-supplied Fortran code. The external function specifies how the grid for the result will be generated. The axes can be inherited from the input arguments or specified in the external function code.

Utility functions, linked in when the external function is compiled, obtain information from Ferret about variables and grids. The utility functions are described in section 6 (p. 227).

Ferret external functions are compiled individually to create shared object (.so) files. These are dynamically linked with Ferret at run time. Ferret looks for shared object files in the directories specified in the FER_EXTERNAL_FUNCTIONS environment variable.

## Ch11 Sec2. GETTING STARTED

Ferret Version 5.0 and later contains everything you need to run the external functions which are included with the distribution. The environment variable FER_EXTERNAL_FUNCTIONS is defined, listing the directory where the shared object files reside. To see a list of the included external functions and their arguments, type

```
> ferret
…
yes? SHOW FUNC/EXTERNAL
Externally defined functions available to Ferret:
ADD_9(A,B,C,D,E,F,G,H,I)
    (demonstration function) adds 9 arguments
AVET(A)
    (demonstration function) returns the time average
    A: data to be time averaged
...
```

## Ch11 Sec2.1. Getting example/development code

To write your own external functions, you will need to get source code and set up a directory in which to work. All of the source code you need to get started (Makefiles, common files, simple examples) can be obtained from the Ferret Home Page. Go to the External Functions page and follow the instructions there.

You will need to download a tar file to get started. When you untar this file you will find that the `ef_utility/` directory contains the `ferret_cmn` subdirectory, containing common files that you need to compile external functions. The `ef_utility/` directory must be in place before you can compile any of the other external function code. The `examples/` directory contains source code and a `Makefile`. You will create further directories with your external functions source code and Makefiles patterned on what is in the exapmles directory.

## Ch11 Sec3. QUICK START EXAMPLE

It's always easier to start coding from an example. Any of the external functions we provide should be documented well enough to serve as a starting point for writing a new function. In this section, we take the most trivial example function, pass_thru, and alter it to do something a little more interesting, if no more useful.

## Ch11 Sec3.1. The times2bad20 function

We'll use the pass_thru(...) function as a template, modifying it into a times2bad20(...) function. This new function will multiply all values by 2.0 and will replace missing value flags with the value 20.0.

Inside any of the example functions, the areas that you need to (are allowed to) modify are set off with

```
c**********************************************************************
*
c*                                              USER CONFIGURABLE PORTION
|
c*
|
c*
V

        ->Insert your code here<-

c*
^
c*
|
c*                                              USER CONFIGURABLE PORTION
|
```

```
c**********************************************************************
*
```

Here's what you need to do to create the new function:

1. move to the `examples/` directory
2. copy `pass_thru.F` to `times2bad20.F`
3. use your favorite editor to change each "pass_thru" to "times2bad20"
4. go down into the "times2bad20_init" section and change the description of the function
5. go to the "times2bad20_compute" subroutine and change the code to look like this

```
c*            result(i,j,k,l) = bad_flag_result
              result(i,j,k,l) = 20

          ELSE

c*            result(i,j,k,l) = arg1(i,j,k,l)
              result(i,j,k,l) = 2 * arg1(i,j,k,l)
```

Assuming you have downloaded all of the `ef_utility/` directory development code and you are still in the `examples/` directory, you should be able to (Figure 11_1)

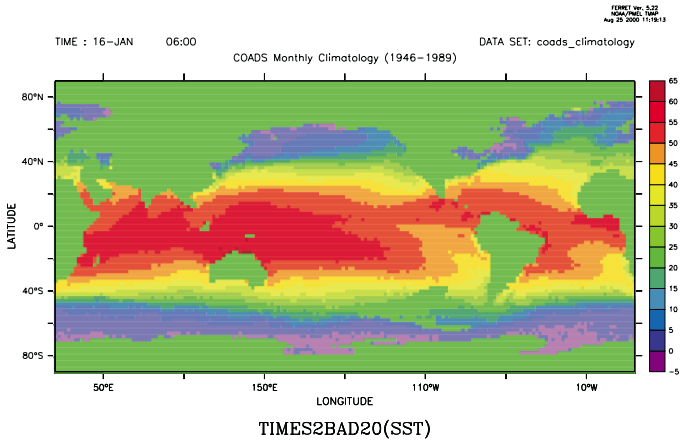



Figure 11_1

```
> make times2bad20.so
> setenv FER_EXTERNAL_FUNCTIONS .
> ferret
  …
yes? use coads_climatology
yes? let a = times2bad20(sst)
yes? shade a[l=1]
```

Congratulations! You have just written your first external function.

## Ch11 Sec4. ANATOMY OF AN EXTERNAL FUNCTION

Every Ferret external function contains an ~_init subroutine which describes the external function's arguments and result grid and a ~_compute subroutine which actually performs the calculation. Three other optional subroutines are available for requesting memory allocation; creating axis limits for the result variable which are extended with respect to the defined region (useful for derivative calculations, etc.); and creating custom axes for the result.

For the following discussion we will assume that our external function is called efname (with source code in a file named efname.F). Examples are also taken from the external functions examples/ directory which you installed when you downloaded the external functions code. This section will briefly describe the work done by the ~_init and ~_compute subroutines. The individual utility functions called by these subroutines are described in the section on Utility Functions below.

When you name your external functions, be aware that Ferret will search its internal function names before the external function names. So if you use a name that is already in use, your function will not be called. Use SHOW FUNCTION from Ferret to list the names that already are in use

## Ch11 Sec4.1. The ~_init subroutine (required)

**subroutine efname_init (id)**

This subroutine specifies basic information about the external function. This information is used when Ferret parses the command line and checks the number of arguments; when it generates the output of SHOW FUNCTION/EXTERNAL; and in determining the result grid.

The following code from examples/subtract.F shows a typical example of an ~_init subroutine. For an example with more arguments please look at examples/add_9.F. For an example where a result axis is reduced with respect to the equivalent input axis take a look at examples/percent_good_t.F.

```
        SUBROUTINE subtract_init(id)

        INCLUDE 'ferret_cmn/EF_Util.cmn'

        INTEGER id, arg

        CALL ef_version_test(ef_version)

*
*************************************************************************
*                                                 USER CONFIGURABLE PORTION
|
*
|
*
V
```

```
      CALL ef_set_desc(id,'(demonstration function) returns: A - B' )

      CALL ef_set_num_args(id, 2)   ! Maximum allowed is 9
      CALL ef_set_axis_inheritance(id, IMPLIED_BY_ARGS,
     .     IMPLIED_BY_ARGS, IMPLIED_BY_ARGS, IMPLIED_BY_ARGS)
      CALL ef_set_piecemeal_ok(id, NO, NO, NO, NO)

      arg = 1
      CALL ef_set_arg_name(id, arg, 'A')
      CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)

      arg = 2
      CALL ef_set_arg_name(id, arg, 'B')
      CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)
*
^
*
|
*                                          USER CONFIGURABLE PORTION
|
*
*******************************************************************

      RETURN
      END
```

## Ch11 Sec4.2. The ~_compute subroutine (required)

**subroutine efname_compute (id, arg_1, arg_2, ..., result, wkr_1, wrk_2, ...)**

This subroutine does the actual calculation. Arguments to the external function and any requested working storage arrays are passed in. Dimension information for the subroutine arguments is obtained from Ferret common blocks in `ferret_cmn/EF_mem_subsc.cmn`. The `mem1lox:mem1hix`, etc. values are determined by Ferret and correspond to the region requested for the calculation.@Body Text = In the `~_compute` subroutine you may call other subroutines which are not part of the efname_compute.F source file.

```
      SUBROUTINE subtract_compute(id, arg_1, arg_2, result)

      INCLUDE 'ferret_cmn/EF_Util.cmn'
      INCLUDE 'ferret_cmn/EF_mem_subsc.cmn'

      INTEGER id

      REAL bad_flag(EF_MAX_ARGS), bad_flag_result
      REAL arg_1(mem1lox:mem1hix, mem1loy:mem1hiy,
     .     mem1loz:mem1hiz, mem1lot:mem1hit)
      REAL arg_2(mem2lox:mem2hix, mem2loy:mem2hiy,
     .     mem2loz:mem2hiz, mem2lot:mem2hit)
      REAL result(memreslox:memreshix, memresloy:memreshiy,
     .     memresloz:memreshiz, memreslot:memreshit)

* After initialization, the 'res_' arrays contain indexing information
* for the result axes.  The 'arg_' arrays will contain the indexing
* information for each variable's axes.
```

```
      INTEGER res_lo_ss(4), res_hi_ss(4), res_incr(4)
      INTEGER arg_lo_ss(4,EF_MAX_ARGS), arg_hi_ss(4,EF_MAX_ARGS),
     .     arg_incr(4,EF_MAX_ARGS)


*
*****************************************************************
*                                          USER CONFIGURABLE PORTION
|
*
|
*
V

      INTEGER i,j,k,l
      INTEGER i1, j1, k1, l1
      INTEGER i2, j2, k2, l2

      CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
      CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
      CALL ef_get_bad_flags(id, bad_flag, bad_flag_result)

      …

*
^
*
|
*                                          USER CONFIGURABLE PORTION
|
*
*****************************************************************
      RETURN
      END
```

Please see the “Loop Indices” section for the example calculation.4.3

## Ch11 Sec4.3. The ~_work_size subroutine (optional)

This routine allows the external function author to request that Ferret allocate memory (working storage) for use by the external function. The memory allocated is passed to the external function when the `~compute` subroutine is called. The working storage arrays are assumed to be REAL*4 arrays; adjust the size of the arrays for other data types. See the sample code under ef_get_coordinates (p. 237) for an example of allocating a REAL*8 work array. The working storage is deallocated after the `~compute` subroutine returns.

When working storage is to be requested, a call to `ef_set_num_work_arrays` must be in the `~init` subroutine:

```
SUBROUTINE efname_init (id)
…
CALL ef_set_num_work_arrays (id,2)
```

A maximum of 9 work arrays may be declared. At the time the ~work_size subroutine is called by Ferret, any of the utility functions that retrieve information from Ferret may be used in the determination of the appropriate working storage size.

Here is an example of a ~work_size subroutine:

```
      SUBROUTINE efname_work_size(id)
      INCLUDE 'ferret_cmn/EF_Util.cmn'
      INCLUDE 'ferret_cmn/EF_mem_subsc.cmn'
      INTEGER id
*
* ef_set_work_array_lens(id, array #, X len, Y len, Z len, T len)
*
      INTEGER nx, ny, id
      INTEGER arg_lo_ss(4,1:EF_MAX_ARGS), arg_hi_ss(4,1:EF_MAX_ARGS),
     .     arg_incr(4,1:EF_MAX_ARGS)
      CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)

      NX = 1 + (arg_hi_ss(X_AXIS,ARG1) - arg_lo_ss(X_AXIS,ARG1))
      NY = 1 + (arg_hi_ss(Y_AXIS,ARG1) - arg_lo_ss(Y_AXIS,ARG1))

      CALL ef_set_work_array_lens(id,1,NX,NY,1,1)
      CALL ef_set_work_array_lens(id,2,NX,NY,1,1)
*

      RETURN
```

In the argument list of the ~compute subroutine, the work array(s) come after the result variable. Declare the workspace arrays using index bounds wrk1lox:wrk2hix, … which were set by the ef_set_work_array_lens call above.

```
      SUBOUTINE efname_compute (arg_1, result, workspace1, workspace2)
…

*   Dimension the work arrays
      REAL workspace1(wrk1lox:wrk1hix, wrk1loy:wrk1hiy,
     .                wrk1loz:wrk1hiz, wrk1lot:wrk1hit)
      REAL workspace2(wrk2lox:wrk2hix, wrk2loy:wrk2hiy,
     .                wrk2loz:wrk2hiz, wrk2lot:wrk2hit)
```

## Ch11 Sec4.4. The ~_result_limits subroutine (optional)

The result limits routine sets the limits on ABSTRACT and CUSTOM axes created by the external function.

An example ~result_limits routine might look like this:

```
      SUBROUTINE my_result_limits(id)
      INCLUDE 'ferret_cmn/EF_Util.cmn'
      INTEGER id, arg, NF
*
      INTEGER arg_lo_ss(4,EF_MAX_ARGS), arg_hi_ss(4,EF_MAX_ARGS),
     .     arg_incr(4,EF_MAX_ARGS)
      INTEGER lo, hi
```

```
CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)

arg = 1
lo = 1
hi = (arg_hi_ss(T_AXIS,arg) - arg_lo_ss(T_AXIS,arg) + 1)/ 2
call ef_set_axis_limits(id, T_AXIS, lo, hi)

RETURN
END
```

## Ch11 Sec4.5. The ~_custom_axes subroutine (optional)

The `~custom_axes` subroutine allows the external function author to create new axes that will be attached the the result of the `~compute` subroutine. An example of such a function might take time series data with a time axis and create, as a result, a Fourier transform with a frequency axis.

The difficulty with the `~custom_axes` subroutine is that not all the Ferret internal information is available to the external function at the time Ferret calls this routine.5

## Ch11 Sec5. NOTES AND SUGGESTIONS

## Ch11 Sec5.1. Inheriting axes

When creating an external function, you can get Ferret to do a lot of conformability checking for you if you "inherit axes" properly. This means that Ferret can be responsible for making sure that the arguments you pass to the function are of the proper dimensionality to be combined together in basic operations such as addition, multiplication etc. For any given axis orientation, X, Y, Z, or, T, two arguments are said to be conformable on that axis if 1) they are either of the same length, or 2) at least one of the arguments has a size of 1 on the axis. ( The terminology "size of 1" may equivalently be thought of as a size of 0. In other words, the data is normal to this axis.) When Ferret encounters a problem it will send an error message rather than passing the data to your external function which might result in a crash.

To get Ferret to do this kind of checking you should inherit axes from as many appropriate arguments as possible. For instance, in subtract.F we have the following sections of code:

```
subtract_init(...)

…
CALL ef_set_axis_inheritance(id, IMPLIED_BY_ARGS,
.     IMPLIED_BY_ARGS, IMPLIED_BY_ARGS, IMPLIED_BY_ARGS)
...
```

This means that the axes of the result, and the index range of the result on those axes, will be determined by arguments.

```
…
arg = 1
CALL ef_set_arg_name(id, arg, 'A')
CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)

arg = 2
CALL ef_set_arg_name(id, arg, 'B')
CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)
...
```

Here we specify that each result axis is dependent upon the axes from both arguments. When Ferret sees this, it knows the arguments must be conformable before it passes them to the external function.

The advantages of this approach are best understood by thinking about this example function "MY_ADD_FUNCTION," which performs a simple addition:

```
LET arg1 = X[x=0:1:.1]
LET arg2 = Y[Y=101:102:.05]
LET my_result = MY_ADD_FUNCTION(arg1, arg2)
```

The desired outcome is that "my_result" is a 2-dimensional field which inherits its X axis from arg1 and its Y axis from arg2.

If arguments and result are on the same grid, you should inherit all axes from all arguments. In general, you should inherit axes from as many arguments as possible.

## Ch11 Sec5.2. Loop indices

**Note: Array indices need not start at 1.**

Because the data passed to an external function is often a subset of the full data set, array indices need not start at 1.

**Note: Indices on separate arguments are not necessarily the same.**

This might occur, for instance, with variables from different data sets.

Because of this, we need to ask Ferret what the appropriate index values are for the result axes and for each axis of each argument. We also need to know whether the increment for each axis of each argument is 0 or 1. An increment of 0 would be returned, for example, as the Y axis increment of an argument which which was only defined on the X axis. The data for this argument would be replicated along the Y axis when needed in a calculation.

The following section of code from subtract.F retrieves the index and increment information:

```
…
CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
```

```
CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
...
```

Once we have this information we must make sure that we don't mix and match indices. It's possible that you can write code which will work in the very simplest cases but will fail when you try something like:

```
yes? let a = my_func(sst[d=1],airt[d=2])
yes? plot a[l=@ave]
```

The solution is straightforward if not very pretty: Assign a separate index to each axis of each argument and index them all separately. The code in `subtract.F` shows how to do it with two arguments:

```
…
i1 = arg_lo_ss(X_AXIS,ARG1)
i2 = arg_lo_ss(X_AXIS,ARG2)
DO 400 i=res_lo_ss(X_AXIS), res_hi_ss(X_AXIS)

   j1 = arg_lo_ss(Y_AXIS,ARG1)
   j2 = arg_lo_ss(Y_AXIS,ARG2)
   DO 300 j=res_lo_ss(Y_AXIS), res_hi_ss(Y_AXIS)

      k1 = arg_lo_ss(Z_AXIS,ARG1)
      k2 = arg_lo_ss(Z_AXIS,ARG2)
      DO 200 k=res_lo_ss(Z_AXIS), res_hi_ss(Z_AXIS)

      l1 = arg_lo_ss(T_AXIS,ARG1)
      l2 = arg_lo_ss(T_AXIS,ARG2)
      DO 100 l=res_lo_ss(T_AXIS), res_hi_ss(T_AXIS)


            IF ( arg_1(i1,j1,k1,l1) .EQ. bad_flag(1) .OR.
     .           arg_2(i2,j2,k2,l2) .EQ. bad_flag(2) ) THEN

               result(i,j,k,l) = bad_flag_result

            ELSE

               result(i,j,k,l) = arg_1(i1,j1,k1,l1) -
     .              arg_2(i2,j2,k2,l2)

            END IF


            l1 = l1 + arg_incr(T_AXIS,ARG1)
            l2 = l2 + arg_incr(T_AXIS,ARG2)
100         CONTINUE

            k1 = k1 + arg_incr(Z_AXIS,ARG1)
            k2 = k2 + arg_incr(Z_AXIS,ARG2)
200      CONTINUE

         j1 = j1 + arg_incr(Y_AXIS,ARG1)
         j2 = j2 + arg_incr(Y_AXIS,ARG2)
300   CONTINUE

      i1 = i1 + arg_incr(X_AXIS,ARG1)
      i2 = i2 + arg_incr(X_AXIS,ARG2)
```

```
 400  CONTINUE
      ...
```

## Ch11 Sec5.3. Reduced axes

For external functions we introduce the concept of "axis reduction." The result of an external function will have axes which are either RETAINED or REDUCED with respect to the argument axes from which they are inherited. By default, all result axes have their axis reduction flag set to RETAINED. Every result axis which has it axis inheritance flag set to IMPLIED_BY_ARGS will have the same extent (context) as the argument axis from which it inherits. Setting the axis reduction flag to REDUCED means that the result axis is reduced to a point by the external function.

The axis reduction flag only needs to be applied when the result is reduced to a point but SET REGION information should still be applied to the external function arguments. (e.g. a function returning a status flag) In such a case the result axes should be IMPLIED_BY_ARGS and REDUCED. (as opposed to NORMAL and RETAINED)

The `percent_good_t.F` function is a good example of where the axis reduction flag needs to be set. This function takes a 4D region of data and returns a time series of values representing the percentage of good data at each time point. Inside the `percent_good_t_init` subroutine we see that the X, Y and Z axes are reduced with respect to the incoming argument:

```
*
*****************************************************************
*                                          USER CONFIGURABLE PORTION
|
*
|
*
V
      CALL ef_set_desc(id,
     . '(demonstration function) returns % good data at each time' )
      CALL ef_set_num_args(id, 1)
      CALL ef_set_axis_inheritance(id, IMPLIED_BY_ARGS,
     .     IMPLIED_BY_ARGS, IMPLIED_BY_ARGS, IMPLIED_BY_ARGS)
      CALL ef_set_axis_reduction(id, REDUCED, REDUCED, REDUCED,
     .     RETAINED)
      CALL ef_set_piecemeal_ok(id, NO, NO, NO, NO)
      arg = 1
      CALL ef_set_arg_name(id, arg, 'A')
      CALL ef_set_arg_desc(id, arg, 'data to be checked')
      CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)
*
^
*
|
*                                          USER CONFIGURABLE PORTION
|
*
*****************************************************************
```

This arrangement allows the user to specify an X/Y/Z region of interest and have this region information used when the argument is passed to the function. If we had specified X/Y/Z as NORMAL axes, Ferret would have understood this to mean that all region information for these three axes can be ignored when the percent_good_t function is called. This is not what we want.

## Ch11 Sec5.4. String Arguments

Ferret can pass strings to external functions. This may be useful if you are writing external functions to write a new output format, for example, and wish to pass the output filename as an argument.

By default, all arguments are assumed to be of type `FLOAT_ARG`. In the `~init` subroutine, the external function must tell Ferret which arguments are to be handled as strings:

```
arg = 1
CALL ef_set_arg_type(id, arg, STRING_ARG)
CALL ef_set_arg_name(id, arg, 'message')
CALL ef_set_arg_desc(id, arg, 'String to be written when executing.')
CALL ef_set_axis_influence(id, arg, YES, YES, YES, YES)
```

In the `~compute` subroutine, a pointer to the string argument is passed in and dimensioned as any other argument. A text variable must be declared and a utility function is used to get the actual text string. As an example:

```
      SUBROUTINE string_args_compute(id, arg_1, arg_2, result)

      INCLUDE 'ferret_cmn/EF_Util.cmn'
      INCLUDE 'ferret_cmn/EF_mem_subsc.cmn'

      INTEGER id

      REAL bad_flag(1:EF_MAX_ARGS), bad_flag_result
      REAL arg_1(mem1lox:mem1hix, mem1loy:mem1hiy,
     .           mem1loz:mem1hiz, mem1lot:mem1hit)

      REAL arg_2(mem2lox:mem2hix, mem2loy:mem2hiy,
     .           mem2loz:mem2hiz, mem2lot:mem2hit)
      REAL result(memreslox:memreshix, memresloy:memreshiy,
     .            memresloz:memreshiz, memreslot:memreshit)

      INTEGER res_lo_ss(4), res_hi_ss(4), res_incr(4)
      INTEGER arg_lo_ss(4,1:EF_MAX_ARGS), arg_hi_ss(4,1:EF_MAX_ARGS),
     .     arg_incr(4,1:EF_MAX_ARGS)

      CHARACTER arg1_text*160

*
*************************************************************************
*                                                 USER CONFIGURABLE PORTION
|
*
|
*
```

```
V
      INTEGER i,j,k,l
      INTEGER i1, j1, k1, l1

      CALL ef_get_arg_string(id, 1, arg1_text)

      WRITE(6,49) arg1_text
 49   FORMAT ('The text for arg1 is : ''',a,'''')

      …
```

## Ch11 Sec6. UTILITY FUNCTIONS

The lists below describe the utility functions built into Ferret which are available to the external function writer. These are used to set parameters associated with the external function and to retrieve information provided by Ferret. (Input variables, sending information to Ferret, are in plain type and output variables, getting information from Ferret, are in italic.)

## Ch11 Sec6.1. EF_Util.cmn

External functions need to include the `EF_Util.cmn` file in each subroutine in order to use various pre-defined parameters. These parameters are defined in the table below:

Parameters defined in `EF_Util.cmn`

**To make the code more readable:**

| | | | |
|---|---|---|---|
| X_AXIS (=1) | ARG1 (=1) | ARG5 (=5) | ARG9 (=9) |
| Y_AXIS (=2) | ARG2 (=3) | ARG6 (=6) | YES (=2) |
| Z_AXIS (=3) | ARG3 (=3) | ARG7 (=7) | NO (=0) |
| T_AXIS (=4) | ARG4 (=4) | ARG8 (=8) | |

**Internal parameters for Ferret:**

| | |
|---|---|
| CUSTOM | result axis is defined by the external function |
| IMPLIED_BY_ARGS | result axis is inherited from one (or more) of the arguments |
| NORMAL | this axis does not exist in the result |
| ABSTRACT | result axis is an indexed axis [1:N] |
| RETAINED | result axis has same extent as argument axis |
| REDUCED | result axis is reduced to a point |

## Ch11 Sec6.2. Available utility functions

### Setting Parameters

General Information

- ef_set_desc(id, desc) (p. 229)
- ef_set_num_args(id, num) (p. 229)
- ef_set_piecemeal_ok(id, Xyn, Yyn, Zyn, Tyn) (p. 230)
- ef_set_axis_inheritance(id, Xsrc, Ysrc, Zsrc, Tsrc) (p. 229)
- ef_set_arg_name(id, arg, name) (p. 230)
- ef_set_arg_desc(id, arg, desc) (p. 230)
- ef_set_arg_unit(id, arg, unit) (p. 230)
- ef_set_arg_type(id, arg, type) (p. 231)
- ef_set_axis_influence(id, arg, Xyn, Yyn, Zyn, Tyn) (p. 231)
- ef_set_axis_reduction(id, Xred, Yred, Zred, Tred) (p. 232)
- ef_set_axis_extend(id, arg, axis, lo_amt, hi_amt) (p. 231)
- ef_set_axis_limits(id, axis, lo, hi) (p. 232)
- ef_set_custom_axis(id, axis, lo, hi, delta, unit, modulo) (p. 232)
- ef_set_num_work_arrays(id, num) (p. 233)
- ef_set_work_array_dims(id, array, Xlo, Ylo, Zlo, Tlo, Xhi, Yhi, Zhi, Thi) (p. 233)

### Getting Information

For all calculations

- ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr) (p. 233)
- ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr) (p. 235)
- ef_get_bad_flags(id, bad_flag, bad_flag_result) (p. 236)

Text

- ef_get_arg_info(id, arg, name, title, units) (p. 234)
- ef_get_arg_string(id, arg, text) (p. 234)
- ef_get_axis_info(id, arg, name, units, bkwd, modulo, regular) (p. 235)
- ef_get_axis_dates(id, arg, tax, numtimes, datebuf) (p. 235)

Values

- Ef_get_arg_ss_extremes(id, arg, ss_min, ss_max) (p. 236)
- ef_get_coordinates(id, arg, axis, lo, hi, coords) (p. 237)
- ef_get_box_size(id, arg, axis, lo, hi, size) (p. 238)
- ef_get_box_limits(id, arg, axis, lo, hi, lo_lims, hi_lims) (p. 239)
- ef_get_one_val(id, arg, value) (p. 240)

### Other

- ef_version_test(version) (p. 240)
- ef_bail_out(id, text) (p. 240)

## ef_set_desc(id, desc)

Assign a text string description to the external function.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **CHARACTER*(*) desc:** description of this function

## ef_set_num_args(id, num)

Specify the number of arguments this function will accept. The maximum number of arguments allowed is 9

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER num:** number of arguments for this function

## ef_set_axis_inheritance(id, Xsrc, Ysrc, Zsrc, Tsrc)

Specify where the result axes will come from. The acceptable values for each axis will be one of:

| | |
|---|---|
| CUSTOM | result axis is defined by the external function |
| IMPLIED_BY_ARGS | result axis is inherited from one (or more) of the arguments |
| NORMAL | this axis does not exist in the result |
| ABSTRACT | result axis is an indexed axis [1:N] |

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER Xsrc:** inheritance flag for the X axis
3. **INTEGER Ysrc:** inheritance flag for the Y axis
4. **INTEGER Zsrc:** inheritance flag for the Z axis
5. **INTEGER Tsrc:** inheritance flag for the T axis

## ef_set_piecemeal_ok(id, Xyn, Yyn, Zyn, Tyn)

Tell Ferret whether it is ok to break up calculations along a particular axis. (Not implemented as of Ferret v5.22, EF version 1.3)

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER Xyn:** yes/no flag for the X axis
3. **INTEGER Yyn:** yes/no flag for the Y axis
4. **INTEGER Zyn:** yes/no flag for the Z axis
5. **INTEGER Tyn:** yes/no flag for the T axis

## ef_set_arg_name(id, arg, name)

Assign a text string name to an argument.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **CHARACTER*(*) name:** argument name

## ef_set_arg_desc(id, arg, desc)

Assign a text string description to an argument.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **CHARACTER*(*) desc:** argument description

## ef_set_arg_unit(id, arg, unit)

Assign a text string to an argument's units.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **CHARACTER*(*) unit:** unit description

## ef_set_arg_type(id, arg, type)

Specify the type of an argument as either FLOAT_ARG or STRING_ARG. In the ~_compute subroutine, the ef_get_arg_string() function is used to obtain the desired text string.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **INTEGER type:** either FLOAT_ARG or STRING_ARG

## ef_set_axis_extend(id, arg, axis, lo_amt, hi_amt)

Tell Ferret to extend the range of data passed for an argument. This is useful for cases like smoothers where the result at a particular point depends upon a range of input values around that point.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **INTEGER axis:** axis number
4. **INTEGER lo_amt:** extension to the lo range (–1 means get one more point than in the result)
5. **INTEGER hi_amt:** extension to the hi range (+1 means get one more point than in the result)

## ef_set_axis_influence(id, arg, Xyn, Yyn, Zyn, Tyn)

Specify whether this argument's axes "influence" the result axes. A value of YES for a particular axis means that the result should have the same axis as this argument. If the result should have the same axis as several input arguments, then each argument should specify YES for the axis in question. Note that ef_set_axis_inheritance must have specified IMPLIED_BY_ARGS for this axis.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **INTEGER Xyn:** influence flag for the X axis
4. **INTEGER Yyn:** influence flag for the Y axis
5. **INTEGER Zyn:** influence flag for the Z axis
6. **INTEGER Tyn:** influence flag for the T axis

## ef_set_axis_reduction(id, Xred, Yred, Zred, Tred)

Specify whether the result axes are RETAINED or REDUCED with respect to the argument axes from which they are inherited. Setting the axis reduction flag to REDUCED means that the result axis is reduced to a point by the external function. The axis reduction flag need only be set when the result is reduced to a point but SET REGION information should still be applied to the external function arguments.

Input arguments:
1. **INTEGER id**: external function's ID number
2. **INTEGER Xred**: reduction flag for the X axis
3. **INTEGER Yred**: reduction flag for the Y axis
4. **INTEGER Zred**: reduction flag for the Z axis
5. **INTEGER Tred**: reduction flag for the T axis

## ef_set_axis_limits(id, axis, lo, hi)

Specify the lo and hi limits of an axis. (This is not needed for most functions and must appear in a separate subroutine named `~func_name~_result_limits(id)`).

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER axis:** axis number
3. **INTEGER lo:** index value of the lo range of this axis
4. **INTEGER hi:** index value of the hi range of this axis

## ef_set_custom_axis(id, axis, lo, hi, delta, unit, modulo)

Create a custom axis. This is only used by functions which create a custom axis and must appear in a separate subroutine named `~func_name~_custom_axes(id)`.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER axis:** axis number
3. **INTEGER lo:** index value of the lo range of this axis
4. **INTEGER hi:** index value of the hi range of this axis
5. **INTEGER delta:** increment for this axis
6. **CHARACTER*(*) unit:** unit for this axis
7. **INTEGER modulo:** flag for modulo axes (1 = modulo)

## ef_set_num_work_arrays(id, nwork)

Set the number of work arrays to be allocated. The maximum number of work arrays allowed is 9.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER nwork:** number of storage arrays

## ef_set_work_array_dims(id, iarray, xlo, ylo, zlo, tlo, xhi, yhi, zhi, thi)

Set the working array axis lengths.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER iarray:** array number
3. **INTEGER xlo:** index value of the lo range of x axis
4. **INTEGER ylo:** index value of the lo range of y axis
5. **INTEGER zlo:** index value of the lo range of z axis
6. **INTEGER tlo:** index value of the lo range of t axis
7. **INTEGER xhi:** index value of the hi range of x axis
8. **INTEGER yhi:** index value of the hi range of y axis
9. **INTEGER zhi:** index value of the hi range of z axis
10. **INTEGER thi:** index value of the hi range of t axis

## ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)

Return lo and hi indices and increments to be used in looping through the calculation of the result.

Input arguments:
1. **INTEGER id:** external function's ID number

Output arguments:
1. **INTEGER res_lo_ss(4):** the lo end indices for the X, Y, Z, T axes of the result
2. **INTEGER res_hi_ss(4):** the hi end indices for the X, Y, Z, T axes of the result
3. **INTEGER res_incr(4):** the increment to be applied to the X, Y, Z, T axes of the result

Sample code:

```
CALL ef_get_res_subscripts(id,
res_lo_ss, res_hi_ss, res_incr) ... DO 400 i=res_lo_ss(X_AXIS),
res_hi_ss(X_AXIS) DO 300 j=res_lo_ss(Y_AXIS), res_hi_ss(Y_AXIS)
... 300 CONTINUE 400 CONTINUE
```

## ef_get_arg_info(id, iarg, arg_name, arg_title, arg_units)

Return strings describing argument: name, title, units.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER iarg:** argument number

Output arguments:
1. **CHARACTER*24 arg_name:** the name of the argument
2. **CHARACTER*128 arg_title:** title associated with the argument
3. **CHARACTER*32 arg_units:** the argument's units.

## ef_get_arg_string(id, iarg, text)

Return the string associated with an argument of type STRING_ARG. The maximum length for the string ist 160 characters.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER iarg:** argument number

Output arguments:
1. **CHARACTER*(*) text:** the actual text string for the argument

Sample code:

```
      …

      CHARACTER arg_text*160

*
*************************************************************************
*                                              USER CONFIGURABLE PORTION
|
*
|
*
V
      INTEGER i,j,k,l
      INTEGER i1, j1, k1, l1

      CALL ef_get_arg_string(id, 1, arg_text)
```

```
      WRITE(6,49) arg_text
 49   FORMAT ('The text is : ''',a,'''')

      …
ef_get_axis_info(id, iarg, axname, ax_units, backward, modulo, regular)
```

Return strings describing argument: name, title, units.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER iarg:** argument number

Output arguments:
1. **CHARACTER*16 ax_name(4):** the name of the four axes
2. **CHARACTER*16 ax_units(4):** units of the four axes
3. **LOGICAL backward(4):** true if axis is backward axis
4. **LOGICAL modulo(4):** true if axis is modulo axis
5. **LOGICAL regular(4):** true if axis is regular axis

## ef_get_axis_dates(id, iarg, taxis, numtimes, datebuf)

Returns the string date buffer associated with the time axis of an argument.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER iarg:** argument number
3. **REAL*8 taxis(numtimes):** time axis coordinate values
4. **INTEGER numtimes:** number of time

Output arguments:
1. **CHARACTER*20 datebuf(numtimes):** the string-date buffer for each time.

## ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)

Return lo and hi indices and increments to be used in looping through the calculation of the result.

Input arguments:
1. **INTEGER id:** external function's ID number

Output arguments:
1. **INTEGER arg_lo_ss(4,EF_MAX_ARGS):** the lo end indices for the X, Y, Z, T axes of each argument

2. **INTEGER arg_hi_ss(4,EF_MAX_ARGS):** the hi end indices for the X, Y, Z, T axes of each argument
3. **INTEGER arg_incr(4,EF_MAX_ARGS):** the increment to be applied to the X, Y, Z, T axes of each argument

Sample code:

```
INTEGER i,j,k,l
INTEGER i1, j1, k1, l1
INTEGER i2, j2, k2, l2

CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
CALL ef_get_bad_flags(id, bad_flag, bad_flag_result)

i1 = arg_lo_ss(X_AXIS,ARG1)
i2 = arg_lo_ss(X_AXIS,ARG2)

DO 400 i=res_lo_ss(X_AXIS), res_hi_ss(X_AXIS)
  …
  i1 = i1 + arg_incr(X_AXIS,ARG1)
  i2 = i2 + arg_incr(X_AXIS,ARG2)
400 CONTINUE
```

## ef_get_arg_ss_extremes(id, arg, ss_min, ss_max)

Return the maximum and minim index values for all the arguments. These define the domain of the data.

Input arguments:
1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number

Output arguments:
1. **INTEGER ss_min(4,EF_MAX_ARGS):** the minimum indices for the X, Y, Z, T axes of each argum
2. **INTEGER ss_max(4,EF_MAX_ARGS):** the maximum indices for the X, Y, Z, T axes of each argum

Sample code:

```
CALL ef_get_arg_ss_extremes(id, arg, ss_min, ss_max)
```

## ef_get_bad_flags(id, bad_flag, bad_flag_result)

Return the missing value flags for each argument and for the result.

Input arguments:
1. **INTEGER id:** external function's ID number

Output arguments:
1. **`REAL bad_flag(EF_MAX_ARGS):`** missing value flags for each argument
2. **`REAL bad_flag_result:`** missing value flag for the result

Sample code:

```
CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
CALL ef_get_bad_flags(id, bad_flag, bad_flag_result)

…

IF ( arg_1(i1,j1,k1,l1) .EQ. bad_flag(ARG1) ) THEN
  result(i,j,k,l) = bad_flag_result
ELSE
  ...
```

## ef_get_coordinates(id, arg, axis, lo, hi, coords)

Return the "world coordinates" associated with a particular arg, axis and lo:hi range.

Input arguments:
1. **`INTEGER id:`** external function's ID number
2. **`INTEGER arg:`** argument number
3. **`INTEGER axis:`** axis number
4. **`INTEGER lo:`** lo index of desired range
5. **`INTEGER hi:`** hi index of desired range

Output arguments:

1. **`REAL*8 coords(*):`** array of "world coordinate" values (NB_ these values are associated with index values lo:hi but are returned as `coords(1:hi-lo)`.)

Sample code: in the **work_size**subroutine, define twice as many elements as coordinates so as to have storage for REAL*8 numbers

```
*
      SUBROUTINE myfcn_work_size(id)
      INCLUDE 'ferret_cmn/EF_Util.cmn'
      INCLUDE 'ferret_cmn/EF_mem_subsc.cmn'
      INTEGER id

*  Set the work arrays,  X/Y/Z/T dimensions

      INTEGER nxout, nx2
      INTEGER arg_lo_ss(4,1:EF_MAX_ARGS), arg_hi_ss(4,1:EF_MAX_ARGS),
          arg_incr(4,1:EF_MAX_ARGS)

      CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)

      nxout = 1 + arg_hi_ss(X_AXIS,ARG4) - arg_lo_ss(X_AXIS,ARG4)
```

```
      nx2 = nxout* 2

* Define work array XAX

      CALL ef_set_work_array_dims (id, 1, 1, 1, 1, 1, nx2, 1, 1, 1)
      RETURN
      END
```

In the **compute**subroutine, dimension the REAL*8 coordinate array with half the **wkr1hix**dimension (wrk1lox:wrk1hix, etc are defined by the **work_size**subroutine)

```
      SUBROUTINE myfcn_compute(id, arg_1, arg_2, result,  xax)
…
      REAL arg_1(mem1lox:mem1hix, mem1loy:mem1hiy, mem1loz:mem1hiz,
     .     Mem1lot:mem1hit)
      REAL result(memreslox:memreshix, memresloy:memreshiy,
     .     memresloz:memreshiz, memreslot:memreshit)

      INTEGER res_lo_ss(4), res_hi_ss(4), res_incr(4)
      INTEGER arg_lo_ss(4,EF_MAX_ARGS), arg_hi_ss(4,EF_MAX_ARGS),
     .     Arg_incr(4,EF_MAX_ARGS)

C  Dimension the work array: X dimension was  defined twice as large
C  as the # coordinates, for double precision work array.

      REAL*8 xax(wrk1lox:wrk1hix/2, wrk1loy:wrk1hiy,
     .     wrk1loz:wrk1hiz, wrk1lot:wrk1hit)
…

      CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
      CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
      CALL ef_get_bad_flags(id, bad_flag, bad_flag_result)

      CALL ef_get_coordinates(id, ARG1, X_AXIS, arg_lo_ss(X_AXIS,
     .  ARG1), arg_hi_ss(X_AXIS, ARG1), xax )

…

     dummy = 1
     DO 30 i = arg_lo_ss(Y_AXIS, ARG1), arg_hi_ss(Y_AXIS, ARG1)
       cstr(i) = 1.0 / cos( xax(dummy) * (1.0/radian) )
       dummy = dummy + 1
30   CONTINUE
```

## ef_get_box_size(id, arg, axis, lo, hi, size)

Return the box sizes (in “world coordinates”) associated with a particular arg, axis and lo:hi range.

Input arguments:

1. **INTEGER id:** external function’s ID number
2. **INTEGER arg:** argument number
3. **INTEGER axis:** axis number
4. **INTEGER lo:** lo index of desired range
5. **INTEGER hi:** hi index of desired range

Output arguments:

1. **REAL size(*):** array of box size values (NB_ these values are associated with index values lo:hi but are returned as `coords(1:hi-lo)`.)

Sample code:

```
REAL tk_y(wrk1lox:wrk1hix, wrk1loy:wrk1hiy/2,
     .     wrk1loz:wrk1hiz, wrk1lot:wrk1hit)
REAL tk_dx(wrk2lox:wrk2hix, wrk2loy:wrk2hiy,
     .     wrk2loz:wrk2hiz, wrk2lot:wrk2hit)

INTEGER dummy

…

CALL ef_get_res_subscripts(id, res_lo_ss, res_hi_ss, res_incr)
CALL ef_get_arg_subscripts(id, arg_lo_ss, arg_hi_ss, arg_incr)
CALL ef_get_bad_flags(id, bad_flag, bad_flag_result)
CALL ef_get_coordinates(id, ARG1, Y_AXIS, arg_lo_ss(Y_AXIS, ARG1),
.     arg_hi_ss(Y_AXIS, ARG1), tk_y )
CALL ef_get_box_size(id, ARG1, X_AXIS, arg_lo_ss(X_AXIS, ARG1),
.     arg_hi_ss(X_AXIS, ARG1), tk_dx )

…

dummy = 1
DO 20 i = arg_lo_ss(X_AXIS, ARG1), arg_hi_ss(X_AXIS, ARG1)
  dxt4r(i) = 1.0 / ( 4.0 * tk_dx(dummy) * radius/radian )
  dummy = dummy + 1
20 CONTINUE
```

## ef_get_box_limits(id, arg, axis, lo, hi, lo_lims, hi_lims)

Return the box limits (in "world coordinates") associated with a particular arg, axis and lo:hi range.

Input arguments:

1. **INTEGER id:** external function's ID number
2. **INTEGER arg:** argument number
3. **INTEGER axis:** axis number
4. **INTEGER lo:** lo index of desired range
5. **INTEGER hi:** hi index of desired range

Output arguments:

1. **REAL lo_lims(*):** array of box lower limit values (NB_ these values are associated with index values lo:hi but are returned as `coords(1:hi-lo)`.)
2. **REAL hi_lims(*):** array of box upper limit values (NB_ these values are associated with index values lo:hi but are returned as `coords(1:hi-lo)`.)

## ef_get_one_val(id, arg, value)

Return the value of 1×1×1×1 variable.

Input arguments:
1. **INTEGER id**: external function's ID number
2. **INTEGER arg**: argument number

Output arguments:
1. **REAL *value*** : The value of the variable

## ef_version_test (version)

Return the version number of the external functions code that is in place.

Output argument:
1. **REAL *version*** : The version number

## ef_bail_out(id, text)

Bail out of an external function, returning to Ferret and issuing a message to the user.

Input arguments:
1. **INTEGER id**: external function's ID number
2. **INTEGER text**: text string to output.

# Part II: COMMANDS REFERENCE

## Ref Sec1. ALIAS

An alias for DEFINE ALIAS.

## Ref Sec2. CANCEL

Cancels a program state or definition—generally paired with a SET or DEFINE command. See commands SET (p. 288) and DEFINE (p. 251) for further information.

**Arguments:**

The arguments, which are names of variables, data sets, or other definitions can be specified using wildcards. The * wildcard matches any number of characters in the name; the ? wildcard matches exactly one character.

## Ref Sec2.1. CANCEL ALIAS

Cancels a user-defined command alias.

```
yes? CANCEL ALIAS ALIAS_NAME
```

The command UNALIAS is an alias for CANCEL ALIAS.

## Ref Sec2.2. CANCEL AXIS

/MODULO

CANCEL AXIS forms the complement to DEFINE AXIS. It is also applicable to "persistent" axes which are defined by netCDF files such as climatological_axes.cdf -- axes which are not associated with any variables in the netCDF file, itself, and are not automatically deleted when the data set is canceled.

Attempts to CANCEL AXIS on a axis which is used by a variable in a currently open data set will be rejected with a message indicating the reason.

**Command qualifiers for CANCEL AXIS:**

CANCEL AXIS/**MODULO**
Cancels the modulo nature of a user-defined axis.

```
yes? CANCEL AXIS/MODULO my_x_axis
```

or

```
yes? CANCEL AXIS/MODULO my_t*
```

## Ref Sec2.3. CANCEL DATA_SET

/ALL /NOERROR

Removes the specified data set from the list of available sets.

```
yes? CANCEL DATA_SET dset1, dset2, ..., dsetn
```
where each dset may be the name or number of a data set; or
```
yes? CANCEL DATA/ALL
```

(See also SET DATA_SET, p. 289, and SHOW DATA SET, p. 318.)

**Command qualifiers for CANCEL DATA_SET:**

CANCEL DATA/**ALL**
Eliminates all data sets from the list of accessible data sets.

CANCEL DATA/**NOERROR**
Suppresses the error message otherwise generated when a data set that was never set is canceled. Useful in GO scripts for closing data sets that may have been opened in previous usage of the script.

Note that if a grid or axis from a netCDF file is used in the definition of a LET-defined variable (e.g. LET my_X = X[g=sst[D=coads_climatology]]) that variable definition will be invalidated when the data set is canceled (CANCEL DATA coads_climtology, in the preceding example). There is a single exception to this behavior: netCDF files such as climtological_axes.cdf, which define grids or axes that are not actually used by any variables. These grids and axes will remain defined even after the data set, itself, has been canceled. They may be deleted with explicit use of CANCEL GRID or CANCEL AXIS.

## Ref Sec2.4. CANCEL EXPRESSION

Un-specifies the current context expression. Ferret's "action" commands can be issued without an argument (e.g., `yes? PLOT`), in which case Ferret uses the current context expression. This expression is either the argument of the most recent action command, or an expression set explicitly with SET EXPRESSION.

```
yes? CANCEL EXPRESSION
```

The qualifier /ALL can be used with this command, but it exists for compatibility purposes only and has no effect.

## Ref Sec2.5. CANCEL GRID

CANCEL GRID forms the complement to DEFINE GRID It is also applicable to "persistent" grids which are defined by netCDF files such as climatological_axes.cdf -- grids which are not associated with any variables in the netCDF file, itself, and are not automatically deleted when the data set is canceled.

Attempts to CANCEL GRID on a grid or axis which is used by a variable in a currently open data set will be rejected with a message indicating the reason.

## Ref Sec2.6. CANCEL LIST

/ALL /APPEND /FILE /FORMAT /HEADING /PRECISION

Toggles the effects of the SET LIST command. See command SET LIST (p. 296).

```
yes? CANCEL LIST[/qualifiers]
```

**Command qualifiers for: CANCEL LIST**

CANCEL LIST/**ALL**
Restores all aspects of the LIST command to their default behavior.

CANCEL LIST/**APPEND**
Resets the listed output to NOT append to existing file.

CANCEL LIST/**FILE**
Resets the listed output to automatic file naming.

CANCEL LIST/**FORMAT**
Resets the listed output to its default formatting.

CANCEL LIST/**HEAD**
Instructs listed output to omit the descriptive data header.

CANCEL LIST/**PRECISION**
Resets the precision of listed data to 4 significant digits.

## Ref Sec2.7. CANCEL MEMORY

/ALL /PERMANENT /TEMPORARY

Clears data currently cached in memory.

```
yes? CANCEL MEMORY[/qualifier]
```

Use this command to save memory space—by clearing data as soon as it is no longer needed virtual memory requirements can be reduced. This is especially useful for efficient batch processing. Default is CANCEL MEMORY/TEMPORARY.

**Example:**

To produce an animation using minimal virtual memory try:

```
yes? REPEAT/T=lo:hi:delta GO min_mem_movie
```

Where the file min_mem_movie.jnl contains

```
CONTOUR/FRAME  temp[Z=0]        ! contour plot
CANCEL MEMORY/ALL               ! clear memory for next time step
```

**Command qualifiers for CANCEL MEMORY:**

CANCEL MEMORY/**ALL**
Clears all variables stored in memory.

CANCEL MEMORY/**PERMANENT**
Clears all "permanent" variables stored in memory (i.e., variables loaded into memory with LOAD/PERMANENT).

CANCEL MEMORY/**TEMPORARY**(default)
Clears all non-permanent variables stored in memory.

## Ref Sec2.8. CANCEL MODE

Sets the state of a mode to "canceled."

```
yes? CANCEL MODE mode_name
```

(See command SET MODE, p. 299, for descriptions of modes.)

## Ref Sec2.9. CANCEL MOVIE

This command is unnecessary in Ferret version 3.1 and later; it is provided for compatibility with older versions of Ferret. It restores the default movie file name (ferret.mgm) but is not needed to conclude capturing graphics to a movie file.

```
yes? CANCEL MOVIE
```

The qualifier /ALL can be used with this command, but it exists for compatibility purposes only and has no effect.

## Ref Sec2.10. CANCEL SYMBOL

/ALL

Deletes a user-defined symbol (string variable) definition.

```
yes? CANCEL STRING[/qualifier]   [symbol_name]
```

**Command qualifiers for CANCEL SYMBOL:**

CANCEL SYMBOL/**ALL**
Deletes all user-defined symbol definitions.

**Examples:**

```
yes? CANCEL SYMBOL my_x_label   !eliminate my_x_label from the
definitions
yes? CANCEL SYMBOL *x_label     !remove all strings ending in x_label
yes? CANCEL SYMBOL/ALL          !remove all user-defined symbols.
```

## Ref Sec2.11. CANCEL REGION

/I/J/K/L /X/Y/Z/T /ALL

Cancels part or all of the current or named region.

```
yes? CANCEL REGION[/qualifier] [region_name]
```

**Examples:**

```
yes? CANCEL REGION               !clear the current region
yes? CANCEL REGION/T             !eliminate T from the current context
yes? CANCEL REGION reg1          !clear the region named "reg1"
```

**Command qualifiers for CANCEL REGION:**

CANCEL REGION/**I /J /K /L /X /Y /Z /T**
Eliminates I, J, K, L, X, Y, Z, or T axis information from current context or named region.

CANCEL REGION/**ALL**
Eliminates ALL stored region information (rarely used).

## Ref Sec2.12. CANCEL VARIABLE

/ALL /DATASET

Deletes a user-defined variable definition.

```
yes? CANCEL VARIABLE[/qualifier]  [var_name]
```

**Command qualifiers for CANCEL VARIABLE:**

CANCEL VARIABLE/**ALL**
Deletes all user-defined variable definitions.

**Examples:**

```
yes? CANCEL VARIABLE my_sst  !eliminate my_sst from the definitions
yes? CANCEL VARIABLE *wind   !delete all variables ending in wind
yes? CANCEL VARIABLE tau?    !delete variables named tau plus one
character
yes? CANCEL VARIABLE/ALL     !delete all user-defined defined variables
```

CANCEL VARIABLE/**DATASET**
Deletes user define variables associated with the named dataset, which were defined by a DEFINE VARIABLE/DATASET command.

## Ref Sec2.13. CANCEL VIEWPORT

Cancels a defined viewport or cancels use of viewports.

```
yes? CANCEL VIEWPORT view_name        !un-define view_name
yes? CANCEL VIEWPORT                  !return to full window output
```

## Ref Sec2.14. CANCEL WINDOW

/ALL

Removes graphics window(s) from the screen.

```
yes? CANCEL WINDOW  n   !or
yes? CANCEL WINDOW/ALL
```

**Command qualifiers for CANCEL WINDOW:**

CANCEL WINDOW/**ALL**
Removes all graphics windows.

## Ref Sec3. CONTOUR

/I/J/K/L /X/Y/Z/T /D /FILL /FRAME /KEY /LEVELS /LINE /NOAXIS /NOKEY /NOLABEL /OVERLAY /PALETTE /PATTERN /SIZE /SPACING /SIGDIG /PEN /SET_UP /TITLE /COLOR /TRANSPOSE /HLIMITS /VLIMITS /XLIMITS /YLIMITS

Produces a contour plot.

```
yes? CONTOUR[/qualifiers]   [expression]
```

**Example:**

```
yes? CONTOUR var1           !produce a contour plot of the variable
var1
```

**Parameters**

Expressions may be any valid expression. See the chapter “Variables and Expressions”, section “Expressions” (p. 53), for a definition of valid expressions. The expression will be inferred from the current context if omitted from the command line.

**Command qualifiers for CONTOUR:**

CONTOUR/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression being plotted.

CONTOUR/**D=**
Specifies the default data set to use when evaluating the expression being contoured.

CONTOUR/**FILL** (alias FILL)
Creates a color filled contour image.

CONTOUR/**FRAME**
Causes the graphic image produced by the command to be captured as an animation frame in the file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

CONTOUR/**KEY**
Displays a color key for the palette used in a color-filled contour plot. Only valid in conjunction with /FILL (default with CONTOUR/FILL or alias FILL).

CONTOUR/**LEVELS**
Specifies the contour levels or how the levels will be determined. If the /LEVELS qualifier is omitted Ferret automatically selects reasonable contour levels.

See the chapter "Customizing Plots", section "Contouring" (p. 154) for examples and more documentation on /LEVELS and color_thickness indices. See also the demonstration "custom_contour_demo.jnl".

CONTOUR/**LINE**
Overlays contour lines on a color-filled plot. Valid only with /FILL (or as a qualifier to alias FILL). When /LINE is specified the color key, by default, is omitted. Use FILL/LINE/KEY to obtain both contour lines and a color key.

CONTOUR/**NOKEY**
Turns off display of a color key for the palette used in a color-filled contour plot. Only valid in conjunction with /FILL (or with alias FILL).

CONTOUR/**NOAXIS**
Suppresses all axis lines, tics and labeling so that no box appears surrounding the contour plot. This is especially useful for map projection plots.

CONTOUR/**NOLABELS**
Suppresses all plot labels except axis labels.

CONTOUR/**OVERLAY**
Causes the indicated expression to be overlaid on the existing plot.

Note (CONTOUR/OVERLAY with time axes):
A restriction in PPLUS requires that if time is an axis of the contour plot, the overlaid variable must share the same time axis encoding as the base plot variable. If this condition is not met, you may find that the overlaid contour fails to be drawn. The solution is to use the Ferret regridding capability to regrid the base plot variable and the overlaid plot variable onto the same time axis.

CONTOUR/**PALETTE=**
Specifies a color palette (otherwise, the current default palette is used). Valid only with CONTOUR/FILL (or as a qualifier to the alias FILL). The file suffix *.spk is not necessary when specifying a palette. Try the Unix command `% Fpalette '*'` to see available palettes. See command PALETTE (p. 276) for more information.

**Example:**

```
yes? CONTOUR/FILL/PALETTE=land_sea  world_relief
```

The /PALETTE qualifier changes the current palette for the duration of the plotting command and then restores the previous palette. This behavior is not immediately compatible with the /SET_UP qualifier. See the PALETTE (p. 276) command for further discussion.

CONTOUR/**PATTERN=**
Specifies a pattern file (otherwise, the current default pattern specification is used). Valid only with CONTOUR/FILL (or as a qualifier to the alias FILL). The file suffix *.pat is not necessary when specifying a pattern. Try the Unix command % Fpattern '*' to see available patterns. See command PATTERN (p. 277) for more information.

CONTOUR/**COLOR=**
Sets line color (replaces the /PEN qualifier). The available color names are Black, Red, Green, Blue, LightBlue, and , Purple, and White (not case sensitive), corresponding to the /PEN values 1-6, respectively. (/COLOR also accepts numerical values.).

**Example:**

```
yes? CONTOUR/COLOR=red  sst
```

CONTOUR/**PEN=**
Sets line style for contour lines (same arguments as PLOT/LINE=). Argument can be an integer between 1 and 18; run `GO line_samples` to see the styles for color devices.

**Example:**

```
yes? CONTOUR/PEN=2  sst
```

CONTOUR/**SIZE=**
Controls the size of characters in the contour labels, using PLOT+ definition of "inches" .Default is 0.8' See example under **CONTOUR/SPACING=** below.

CONTOUR/**SIGDIG=**
Sets the number of significant digits for contour labels. Default is 2. See example under **CONTOUR/SPACING=** below.

CONTOUR/**SPACING=**

Sets spacing for contour lines, using PLOT+ definition of “inches”. The default spacing is 5.0. (See the CONSET command in the on-line PLOT+ Users Guide)

**Example of CONTOUR/SIZE/SIGDIG/SPACING**

```
yes? LET my_field = SIN(X[x=1:6:.1])*COS(Y[y=1:6:0.1])
yes? CONTOUR/SIGDIG=1/SIZE=0.15/SPACING=3 my_field
```

Specifies contour labels with a single significant digit using characters of height 0.15 "inches" at a nominal spacing of 3 "inches", consistent with the PLOT+ usage of "inches". (These are the same units as in, say, "ppl axlen 8,6", to specify plot axes of lengths 8 and 6 inches for horizontal and vertical axes, respectively.) Note that the PLOT+ CONPRE and CONPST commands are also useful, giving control over the text font and color used in the labels and adding units to the labels. For example, the commands

```
yes? PPL CONPRE @C002@CR
yes? PPL CONPST cm/sec
```

will transform the labels in the above CONTOUR example to red (002), Complex Roman font with a units label of "cm/sec".

CONTOUR/**SET_UP**
Performs all the internal preparations required by program Ferret for contouring but does not actually render output. The command PPL can then be used to make changes to the plot prior to producing output with the PPL CONTOUR command. This permits plot customizations that are not possible with Ferret command qualifiers. See the chapter “Customizing Plots”, section “Contouring” (p. 154).

CONTOUR/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about the expression. To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character.

CONTOUR/**TRANSPOSE**
Causes the horizontal and vertical axes to be interchanged. By default the X and T axes of the data are drawn horizontally on the plot and the Y and Z axes of the data are drawn vertically. For Y-Z plots the Z data axis is vertical by default.

Note that plots in the YT and ZT planes have /TRANSFORM applied by default in order to achieve a horizontal T axis. See /XLIMITS (below) for further details. Use /TRANSPOSE manually to reverse this effect.

CONTOUR/**HLIMITS=**

Specifies axis range and tic interval for the horizontal axis. Without this qualifier, Ferret selects reasonable values.

```
yes? CONTOUR/HLIMITS=lo_val:hi_val[:increment]  [expression]
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

The /HLIMITS and /VLIMITS qualifiers will retain their "horizontal" and "vertical" interpretations in the presence of the /TRANSPOSE qualifier. Thus, the addition of /TRANSPOSE to a plotting command mandates the interchange of "H" and "V" on the limits qualifiers.

CONTOUR/**VLIMITS=**
Specifies the axis range and tic interval for the vertical axis. See /HLIMITS (above)

CONTOUR/**XLIMITS=**
**Note**: **XLIMITS** and **YLIMITS** have been denigrated. Please use **HLIMITS** and **VLIMITS** instead.
Specifies axis range and tic interval for the X axis. Without this qualifier, Ferret selects reasonable values.

```
yes? CONTOUR/XLIMITS=lo_val:hi_val[:increment]  [expression]
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

Note that the “X” in /XLIMITS refers to the horizontal axis of the plot rather than to the X axis of the grid. This can lead to confusion, especially on plots in the YT or ZT plane. Plots in these planes are automatically transposed to place the Y or Z axis, respectively, on the vertical axis of the plot. Plots may also be transposed manually with the /TRANSPOSE qualifier. On transposed plots /XLIMITS will refer to the vertical axis of the plot.

CONTOUR/**YLIMITS=**
**Note**: **XLIMITS** and **YLIMITS** have been denigrated. Please use **HLIMITS** and **VLIMITS** instead.
Specifies the axis range and tic interval for the Y axis. See /XLIMITS (above).

## Ref Sec4. DEFINE

Defines a new alias, region, grid, axis, variable, or viewport.

## Ref Sec4.1. DEFINE ALIAS

Defines an alias for a command. "ALIAS" is an alias for DEFINE ALIAS.

```
yes? DEFINE ALIAS NAME COMMAND
```

**Example:**

```
yes? DEFINE ALIAS SDF SHOW DATA/FULL
```

## Ref Sec4.2. DEFINE AXIS

/X/Y/Z/T /DEPTH /FILE /FROM_DATA /MODULO /NAME /NPOINTS /T0 /UNITS /EDGES

Defines an axis (axis name up to 16 characters).

```
yes? DEFINE AXIS[/qualifiers]  axis_name
```
or
```
yes? DEFINE AXIS[/qualifiers]  axis_name = expr
```

**Example:**

```
yes? DEFINE AXIS/X=140E:140W:.2   AX140
```
or
```
yes? DEFINE AXIS/X   myaxis = {1, 5, 15, 35}
```

**Command qualifiers for DEFINE AXIS:**

DEFINE AXIS/**X=/Y=/Z=/T=**
Specifies the limits and point spacing of an axis.

```
yes? DEFINE AXIS/X=lo:hi:delta   axis_name
```

The limits may be in longitude, latitude, or date format (for X, Y, or T axis, respectively) or may be simple numbers. No units are assumed unless units are given explicitly with the /UNITS qualifier.

Use /UNITS=degrees to obtain latitude or longitude axes. The X or Y qualifier determines which orientation "degrees" refers to.

For T axis, the limits may be dates (dd-mmm-yyyy:hh:mm:ss) or may be time steps. The delta increment is regarded as hours unless the /UNITS qualifier specifies otherwise.

If the time limits are given as dates then this axis produces date-formatted output (unless CANCEL MODE CALENDAR is issued). If the time limits are given as time steps then all instances of this axis are labeled with time step values in the units specified with the /UNITS qualifier.

**Examples (evenly-spaced axes):**

```
yes? DEFINE AXIS/X=140E:140W:.2 ax140
yes? DEFINE AXIS/Y=15S:25N:.5 axynew
yes? DEFINE AXIS/Z=0:5000:20/UNITS=CM/DEPTH axzcm
yes? DEFINE AXIS/T="7-NOV-1953":"23-AUG-1988:11:00":24  axtlife
yes? DEFINE AXIS/T=25:125:5/UNITS=minutes   axt5min
```

DEFINE AXIS/**DEPTH**
Specifies the Z axis to be a depth, positive downward, axis. A depth axis is indicated by a "(-)" following its title in a SHOW GRID or SHOW AXIS command. Depth axes are notated by "UD" (up-down) in the grid definition file, while normal vertical axes (such as an elevation axis in meteorology) are notated by "DU" (down-up).

**Example:**

```
yes? DEFINE AXIS/Z=0:5000:20/DEPTH/UNITS=CM AXZDCM
```

DEFINE AXIS/**EDGES**
The /EDGES qualifier indicates that the coordinates provided refer to the edges or boundaries between grid cells. When /EDGES is used, the coordinates of the grid points will be computed at the midpoints between the indicated edges. When /EDGES is used in conjunction with /FROM_DATA the number of grid points created will be equal to the number of coordinates minus one, since the list of edges includes both the upper and lower edge of the axis. An example of defining an axis by its edges is

```
yes? DEFINE AXIS/Z=0:5010:20/EDGES/DEPTH/UNITS=CM AXZDCM
```

A class of especially important uses for the /EDGES qualifier is to create custom calendar axes. This example creates a true monthly axis, with axis cells beginning on the first of each month:

```
yes? let month = MOD(l-1,12)+1
yes? let add_year = INT((l-1)/12)
yes? let tstep = DAYS1900(1980+add_year,month,1)
yes? define axis/from_data/T/units=days/name=tax/t0=1-jan-1900/edges
tstep[l=1:`20*12+1`]
```

The following example shows the computation of a custom climatological average. Given, for example, a multi-year time series of a daily measured variable, the climatological average of the variable for two unequal time periods could be computed by creating an axis with two points, using the FROM_DATA qualifier. The grid cells for these two points would extend from 15-Mar to 27-May (about 73 days), and from 27-May to 15-Mar (about 292 days). The actual dates on which the 2 points are located would be the midpoints of these two intervals, on 20-apr and 20-oct.

```
yes? define axis/t=1-jan-0001:1-jan-0002:1/unit=days/t0=1-jan-0000
tencoding
yes? let tstep = t[gt=tencoding]
yes? let start_date = tstep[t=15-mar-0001]
yes? let end_date = tstep[t=27-may-0001]
yes? define
axis/from_data/T/units=days/name=tax/t0=1-jan-0000/edges/modulo
{`start_date,p=7`,`end_date,p=7`,`start_date+365.2425,p=7`}
yes? def grid /t=tax taxgrid
yes? sh/l=1:2 grid taxgrid
     GRID TAXGRID
  name       axis              # pts   start                end
  normal     X
  normal     Y
  normal     Z
  TAX        TIME                2mi   20-APR      12:00    20-OCT
02:54
             L     T                 BOX_SIZE       TIME_STEP (DAYS)
             1>  20-APR      12:00:00  73             475.5
             2>  20-OCT      02:54:35  292.2425       658.1212
```

DEFINE AXIS/**FILE=**
Reads a gridfile for grid and axis definitions. The gridfile specified should be in the standard TMAP gridfile format. There are several documents in $FER_DIR/doc regarding gridfiles and TMAP format (e.g., “about_grid_files.txt”).

```
yes? DEFINE AXIS/FILE=grid_file.grd
```

DEFINE AXIS/**FROM_DATA**
Used only in conjunction with /NAME to define an axis from any expression that Ferret can evaluate.

```
yes? DEFINE AXIS/FROM_DATA/NAME=axis_name   expr
```

(This is a mechanism to convert dependent variables into independent axis data.)

When defining an axis from a LET-defined variable or expression the condensed syntax (e.g.)

```
yes? DEFINE AXIS/X axname=expression
```

replaces the older (still supported) syntax

```
yes? DEFINE AXIS/X/NAME=axname/FROM_DATA expression
```

Note that the values from which the axis is to be created must be in strictly increasing order. If the coordinates are repeated, Ferret will “micro-adjust” the values by adding multiples of 1 millionth of the axis range to the repeated values. Ferret will issue an informative message if it is micro-adjusting an axis.

**Example (unevenly-spaced axis):**

```
yes? DEFINE AXIS/FROM_DATA/X/NAME=my_xaxis pos[D=2]^0.5
```

defines each coordinate of the axis “my_xaxis” as the square root of variable “pos” from data set 2.

DEFINE AXIS/**MODULO**
Specifies that the axis being defined be treated as modulo; that is, the first point will wrap around and follow the last point (e.g., a longitude axis).

DEFINE AXIS/**NAME=**
Used only in conjunction with /FROM_DATA to specify the name of the axis to be defined.

```
yes? DEFINE AXIS/FROM_DATA/NAME=axis_name   expr
```

DEFINE AXIS/**NPOINTS=**
Specifies the number of coordinate points on the axis being defined.

```
yes? DEFINE AXIS/Z=lo:hi/NPOINTS=n   ax_name
```

This qualifier can be used instead of specifying `Z=lo:hi:delta`.

DEFINE AXIS/**T0=**
Specifies the date and time associated with the time step value 0.0

**Example:**

```
DEFINE AXIS/T="1-NOV-1980":"15-AUG-1988":72/T0="1-JAN-1800"  TNEW
```

**Note:** The /T0 qualifier is optional; the underlying time step values are transparent to Ferret users for most purposes. The default value is 15-JAN-1901.

DEFINE AXIS/**UNITS=**
Specifies the units of the axis being defined.

A DEFINE AXIS command such as DEFINE AXIS/X=130E:80W:2 xax infers from the formatting of the longitude coordinates the implied qualifier "/UNITS=degrees". Similar for latitudes.

**Example:**

```
yes? DEFINE AXIS/Z=0:2000:100/UNITS=CM   ZCM
```

Any string (up to 10 characters) is acceptable as a units string, but only the following units are recognized and used when computing axis transformations:

| | | |
|---|---|---|
| cm (or centimeter) | mm (or millimeter) | day |
| km (or kilometer) | mb (or millibar) | mon |
| m (or meter) | level | yr (or year) (365 days) |
| deg (or lat or lon) | layer | gregorian_year (365.2425 days) |
| ft (or feet or foot) | sec | year360 (360 days) |
| in | min | year366 (366 days) |
| mile | hour | M2 cycles |
| dbar | mbar | |

**NOTES:**

1) As of Ferret version 5.1 the definition of the unit "month" has been redefined to be exactly 1/12 of a climatological year. This change applies both to files that use "units=months" and to the DEFINE VARIABLE command. The climatological month is the length of an average month in the Gregorian calendar, including leap years -- 1/12 or 365.2485 days. Thus the command

```
yes? DEFINE AXIS/T0=1-JAN-0000/T=0:12:1/EDGES/units=months/MODULO
month_reg
```

defines a climatological monthly axis which does not "drift" over time due to leap years. This non-drift behavior can be observed using a commands like

```
yes? SHOW AXIS /l=1:12001:1200 month_reg
```

which will show every 100th January over 1000 years.

2) The units dbar and mbar are recognized by Ferret, however, no automatic conversion is attempted between these and any other units.

**TIP:**

Ferret will convert recognized units of length to meters and recognized units of time to seconds during transformations such as integration (@IIN and @DIN) and differentiation (@DDB, @DDC, @DDF) (see “General Information about transformations,” p. 76). Using this characteristic it is always possible to query Ferret about the conversion factors from meters or seconds by integrating a grid cell of width one on an axis of the units in question. For example:

```
yes? ! query conversion factor to meters
yes? define axis/x=0:1:1/edges/units=feet xtest  ! 1 point, cell
width=1 unit
yes? let vx = 0*X[gx=xtest]+1                    ! vx = 1
yes? list/prec=7 vx[x=@din]
             0*X[GX=XTEST]+1
             X (FEET): 0 to 1 (integrated)
          0.3048000

yes? ! query conversion factor to seconds
yes? define axis/t=0:1:1/edges/units=month ttest  ! 1 point, cell
```

```
width=1 unit
 *** NOTE: /UNIT=MONTHS is ambiguous ... using 1/12 of 365 days.
yes? let vt = 0*T[gt=ttest]+1                    ! vt = 1
yes? list/prec=7 vt[t=@din]
             0*T[GT=TTEST]+1
             T (MONTH): 0 to 1 (integrated)
          2628000.
```

**Note on DEFINE AXIS:**
Axes which are "in use", because they are used by currently open data sets may now be redefined using DEFINE AXIS.

Previously attempting to redefine an in-use axis generated an error. This feature is especially useful to correct the interpretation of erroneous files, or files which exhibit minor incompatibilities with Ferret. Use this feature with caution as it can be used to "fool" Ferret into an incorrect interpretation of a data file.

## Ref Sec4.3. DEFINE GRID

/X/Y/Z/T /FILE /LIKE

Defines a grid (name may be up to 16 characters).

```
yes? DEFINE GRID[/qualifiers]   grid_name
```

**Example:**

```
yes? DEFINE GRID/LIKE=temp/T=my_t_axis   my_grid
```

**Command qualifiers for DEFINE GRID:**

DEFINE GRID/**X=/Y=/Z=/T=**
Specifies what particular axis is to be the X, Y, Z, or T axis for this grid.

```
yes? DEFINE GRID/X=axname  grid_name
```

The name axname may be the name of an axis, the name of a grid that uses the axis desired, or the name of a variable for which the defining grid uses the axis desired.

For example,

```
yes? DEFINE GRID/X=U   gx
```

will create a grid named gx which is one-dimensional—normal to Y, Z, and T.

**Note:** Many axes possess an orientation implicit in their units, especially latitude, longitude, and time axes. The effects of using an axis in an inappropriate orientation, such as /X=time_axis, are unpredictable.

DEFINE GRID/**FILE=**
Reads a gridfile for GRID and AXIS definitions. The gridfile specified should be in the standard TMAP gridfile format. There are several documents in $FER_DIR/doc regarding gridfiles and TMAP format (e.g., about_grid_files.txt).

**Example:**

```
yes? DEFINE GRID/FILE=new_grids.grd
```

DEFINE GRID/**LIKE=**
Specifies a particular grid (by name or by reference to a variable defined on that grid) to use as a template to create a new grid.

```
yes? DEFINE GRID/LIKE=grid_or_variable_name   grid_name
```

All axes of the grid being created will be identical to the axes of the "LIKE=" grid except those explicitly changed with the /X, /Y, /Z, or /T qualifiers.

**Example:**

```
yes? DEFINE GRID/LIKE=temp[D=2]/Z=ZAX  gnew     !temp from data set 2
```

**Examples: DEFINE GRID**

1) 
```
yes? DEFINE AXIS/T="1-JAN-1980":"31-DEC-1983":24 axday
yes? DEFINE GRID/LIKE=temp/T=axday gday
```
Define grid gday to be like the defining grid for temp but with a 4-year, daily-interval time axis.

2) 
```
yes? DEFINE GRID/LIKE=temp[D=ba022]/T=sst[D=nmc] gnmc3d
```
Define grid gnmc3d like temp from data set ba022 but with the same time axis as sst from data set nmc.

3) 
```
yes? DEFINE AXIS/X=140E:140W:.2 xnew
yes? DEFINE AXIS/Y=5S:5N:.2 ynew
yes? DEFINE AXIS/T="15-FEB-1982":"15-FEB-1984":48 tnew
yes? DEFINE GRID/X=xnew/Y=ynew/T=tnew gnew
```
Define grid gnew from new axes. The grid, gnew, will be normal (perpendicular) to Z.

## Ref Sec4.4. DEFINE REGION

/I/J/K/L /X/Y/Z/T /DI/DJ/DK/DL /DX/DY/DZ/DT /DEFAULT

Defines or redefines a named region_name (first 4 characters are recognized).

```
yes? DEFINE REGION[/qualifiers] region_name
```

If the qualifier /DEFAULT is not given only those axes explicitly named will be stored. If the qualifier /DEFAULT is given all axes will be stored.

**Command qualifiers for DEFINE REGION:**

DEFINE REGION/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies region limits ( =lo:hi or =val).

DEFINE REGION/**DI=/DJ=/DK=/DL=/DX=/DY=/DZ=/DT=**
Specifies a change in region relative to the current settings (=lo:hi or =val). See examples below.

DEFINE REGION/**DEFAULT**
Saves all axes and transformations, not just those explicitly given. Commonly, a GO script begins with "DEFINE REGION/DEFAULT save" and ends with "SET REGION save".

**Examples: DEFINE REGION**

1) `yes? DEFINE REGION/DEFAULT save`
   Stores the current default region under the name "save". The region may be restored at a later time by the command `yes? SET REGION save`.

2) `yes? DEFINE REGION/X xonly`
   Stores the current default X axis limits, only, as region xonly.

3) `yes? DEFINE REGION/DX=-5 xonly`
   Stores the current default X axis limits minus 5 as region xonly.

4) `yes? DEFINE REGION/Y=20S:20N/Z yanz`
   Stores the given limits from the Y axis and the default Z axis limits.

5) `yes? DEFINE REGION/DEFAULT/L=5 l5`
   Stores the current default region with the modification that L, the time step, is stored as 5.

6) `yes? DEFINE REGION/DL=-1:+1 lp2`
   Stores an L region beginning 1 time step earlier and ending 1 time step later than the current default region as region lp2.

## Ref Sec4.5. DEFINE SYMBOL

Allows the user to define a string variable. Symbol names must begin with a letter and contain only letters, digits, underscores, and dollar signs.

```
yes? DEFINE symbol symbol_name=string
```

**Example:**

```
yes? DEFINE symbol my_x_label = sample number
```

## Ref Sec4.6. DEFINE VARIABLE

/D /QUIET /TITLE /UNITS /BAD=

Allows the user to define a variable from a valid algebraic expression. **Note:** LET is an alias for DEFINE VARIABLE.

```
yes? DEFINE VARIABLE[/qualifiers]  name=expression
```

**Example:**

```
yes? LET SPEED = U^2 + V^2
```

**Parameters**

The expression may be any valid expression. See the chapter "Variables and Expressions", section "Expressions" (p. 53) for a definition of valid expressions.

The name specified with DEFINE VARIABLE can be 1 to 24 characters in length—letters, digits, $ and _, beginning with a letter. Pseudo-variable, operator, and function names are reserved and cannot be used (I, J, EQ, SIN,...). See the chapter "Variables and Expressions" (p. 47) for recognized pseudo-variables, operators, and functions.

If the name defined is the same as a variable name in a data set, the user-defined variable is used instead of the file variable. (Look for LET/D=d_set to control this behavior in future Ferret versions.)

To enter expressions in Reverse Polish ordering see SET MODE POLISH (p. 305).

**Examples:**

1) 
```
yes? DEFINE VARIABLE sum = a+b
```
or equivalently
```
yes? LET sum = a+b
```

2) `yes? DEFINE VARIABLE/TITLE="velocity"/UNIT="m/sec" pos[T=@DDC]*0.01`
Defines velocity in m/sec from position, pos, in cm.

**Command qualifiers for DEFINE VARIABLE:**

DEFINE VARIABLE/**BAD=value**

Allows user to control the missing value of user-defined variables The specified value will be used whenever the variable is LISTed (or SAVEd) to a file. Note that the missing value will revert to its default (-1E34) when this variable is combined in further calculations.

**Example:**

```
yes? let/bad=3 gap_3 = I[I=1:5]
yes? list gap_3
              I[I=1:5]
 1   / 1:  1.000
 2   / 2:  2.000
 3   / 3:   ....
 4   / 4:  4.000
 5   / 5:  5.000
yes? let new_var = gap_3 + 5
yes? list new_var
              GAP_3 + 5
 1   / 1:   6.00
 2   / 2:   7.00
 3   / 3:   ....
 4   / 4:   9.00
 5   / 5:  10.00
yes? list/form=(1PG15.3) new_var
              GAP_3 + 5
              X: 0.5 to 5.5
       6.00
       7.00
     -1.000E+34
       9.00
       10.0
```

DEFINE VARIABLE/**D=dataset**

Restricts the scope of the variable name to the named data set. See further discussion in the chapter “Variables and Expressions”, section “Defining New Variables” (p. 99).

The qualifier "DATASET=" (LET/DATASET=...) allows you detailed control over the multiple use of the same name. If the name or number of a data set is supplied then the /dataset qualifier indicates that this variable name is to be defined only in the specified data set. For example

```
yes? LET/dataset=coads_climatology V_geostrophic = SLP[X=@DDC]/(F*RHO)
```

Defines V_geostrophic only in data set coads_climatology. In other data sets the name V_geostrophic may refer to file variables or it may be given different definitions or it may be

undefined. The data set may be specified either by name as in this example or by number as shown by SHOW DATA. Note that variables defined using LET/dataset=[name_or_number] will be shown in the SHOW DATA output for that data set as well as in SHOW VARIABLES.

If the /dataset qualifier is applied without specifying a data set name then the interpretation is different. In this case the named variable becomes a default definition -- one which applies only if a data-set specific variable of the same name does not exist. For example, if the command

```
yes? LET/DATASET sst = temp[Z=0]
```

is issued then sst[D=levitus_climatology] will evaluate to temp[D=levitus_climatology,Z=0] because the variable sst does not exist in levitus_climatology, but sst[D=coads_climatology] will refer to the file variable name sst within the coads_climatology data set.

LET/D is especially useful for editing data sets because it gives a ready way to distinguish between the pre-edit and post-edit versions of the variable. In this example we edit the data set etopo60, replacing a small rectangle in the Pacific Ocean.

**Example:**

```
! Do not use memory-cached data when editing.
! Always reread the most recent version from the file.
yes? SET MODE STUPID
! Save an exact copy of the original data for editing.
! We will call our edited file "new_etopo.cdf"
yes? SET DATA etopo60
yes? LET/D=etopo60 depth = rose
yes? SET VARIABLE/TITLE="edited etopo depth"/UNITS=meters depth
yes? SAVE/FILE=new_etopo.cdf depth
yes? USE new_etopo.cdf

     ! "rose[d=etopo60]"    is the original.
     ! "depth[d=new_etopo]" is the edited version.
     ! Redefine "depth[d=etopo60]" as a tool for for selective editing.
yes? LET/D=etopo60 depth = rose[D=etopo60]-rose[D=etopo60] + correction

     ! An example edit: replace a small region with the value 500
yes? LET correction = 500
yes? SAVE/APPEND/FILE=new_etopo.cdf depth[D=etopo60,X=180:175w,Y=0:2n]
yes? PLOT/X=160e:160w/Y=1n rose[D=etopo60], depth[D=new_etopo]
```

DEFINE VARIABLE/**QUIET**
Suppresses message that, by default, tells you when you are redefining an existing variable. This qualifier is useful in command files. (This is the default behavior starting with Ferret version 5.2)

DEFINE VARIABLE/**TITLE=**
Specifies a title (in quotation marks) for the user-defined variable. This title will be used to label plots and listings. If no title is specified the text of the expression will be used as the title. (See also SET VARIABLE/TITLE, p. 311.)

DEFINE VARIABLE/**UNITS=**
Specifies the units (in quotation marks) of the variable being defined. (See command SET VARIABLE/UNITS, p. 311.)

## Ref Sec4.7. DEFINE VIEWPORT

/CLIP /ORIGIN /SIZE /TEXT /XLIMITS /YLIMITS

Defines a new viewport (a sub-rectangle of the graphics window).

```
yes? DEFINE VIEWPORT[/qualifiers]   view_name
```

Issuing the command SET VIEWPORT is best thought of as entering "viewport mode." While in viewport mode all previously drawn viewports remain on the screen until explicitly cleared with either SET WINDOW/CLEAR or CANCEL VIEWPORT. If multiple plots are drawn in a single viewport without the use of /OVERLAY the current plot will erase and replace the previous one; the graphics in other viewports will be affected only if the viewports overlap. If viewports overlap the most recently drawn graphics will always lie on top, possibly obscuring what is underneath. By default, the state of "viewport mode" is canceled.

**Example:**

```
yes? DEFINE VIEWPORT/XLIMITS=0,.5/YLIMITS=0,.5   LL
```

Defines a viewport that will place graphical output into the lower left quarter of the screen, and names the viewport "LL".

**Command qualifiers for DEFINE VIEWPORT.**

DEFINE VIEWPORT/**CLIP=**
This qualifier is obsolete; see XLIMITS= and /YLIMITS= (below). Specifies the location of the upper right corner of the viewport.

DEFINE VIEWPORT/**ORIGIN=**
This qualifier is obsolete; see /XLIMITS= and /YLIMITS= (below). Specifies the location of the lower left corner of the viewport.

DEFINE VIEWPORT/**SIZE=**
This qualifier is obsolete; see /XLIMITS and /YLIMITS (below). Specifies the scaling factor to use relative to the size of the full window.

DEFINE VIEWPORT/**TEXT=**
Controls shrinkage (or expansion) of text.

```
yes? DEFINE VIEWPORT/TEXT=n  view_name
```

In some cases text appearance may become unacceptable due to viewport size and aspect specifications. A value of 1 produces text of the same size as in the full window; 0 < n < 1 shrinks the text; n > 1 enlarges text. Sensible values go up to about 2. When the qualifier /TEXT is omitted, Ferret computes a text size that is appropriate to the size of the viewport.

Note that /TEXT modifies the prominence of the text through manipulation of axis lengths rather than through direct manipulation of the many text size specifications. A low value of text prominence produces axes that are "long" (as seen with SHOW SYMBOLS, p. 167, or PPL LIST XAXIS, p. 136), making the (fixed size) text appear less prominent.

DEFINE VIEWPORT/**XLIMITS=/YLIMITS=**
Specifies the portion of the full window to be used.

```
yes? DEFINE VIEWPORT/XLIMITS=x1,x2/YLIMITS=y1,y2   view_name
```

The values of the limits must be in the range [0,1]; they refer to the portion of the window (of height and length 1) which defines the viewport. Together, /XLIMITS and /YLIMITS replace the CLIP, ORIGIN, and SIZE qualifiers in older Ferret versions.

## Ref Sec5. ELIF

The ELIF command is a part of Ferret's conditional command execution capability: IF-THEN-ELIF-ELSE-ENDIF. It is valid only inside of an IF block. See further description under the IF command (p. 267) in this Commands Reference section.

## Ref Sec6. ELSE

The ELSE command is a part of Ferret's conditional command execution capability: IF-THEN-ELIF-ELSE-ENDIF. It is valid only inside of an IF block. See further description under the IF command (p. 267) in this Commands Reference section.

## Ref Sec7. ENDIF

The ENDIF command is a part of Ferret's conditional command execution capability: IF-THEN-ELIF-ELSE-ENDIF. It is valid only inside of an IF block. See further description under the IF command (p. 267) in this Commands Reference section.

## Ref Sec8. EXIT

/COMMAND_FILE

When issued interactively this command terminates program Ferret.

When executed within a command file this command terminates the execution of the command file and returns control to the level in Ferret that executed the file (the user or another command file).

**Command qualifiers for EXIT:**

EXIT/**COMMAND_FILE**
When executed from within a command file EXIT/COMMAND_FILE forces an immediate exit from Ferret rather than returning control to the user or another command file.

## Ref Sec9. FILE

The FILE command is an alias for SET DATA/EZ (p.293 ). All qualifiers and restrictions are identical to SET DATA/EZ

**Example:**

```
yes? FILE/VARIABLES="u,v" velocities.dat
```

is equivalent to

```
yes? SET DATA/EZ/VARIABLES="u,v" velocities.dat
```

## Ref Sec10. FILL

Alias for CONTOUR/FILL (color-filled contour plot). All qualifiers and restrictions are identical to CONTOUR/FILL.

**Example:**

```
yes? FILL/PAL=land_sea  etopo60
```

is equivalent to

```
yes? CONTOUR/FILL/PAL=land_sea  etopo60
```

## Ref Sec11. FRAME

/FORMAT /FILE

Saves the current graphics display image as a frame in the movie file initialized with the command SET MOVIE. FRAME is also a qualifier for the “action” commands PLOT, CONTOUR, POLYGON, SHADE, VECTOR and WIRE.

```
yes? CONTOUR my_var
yes? FRAME
```

Note that FRAME follows a command which creates an image.

FRAME/**FORMAT=format** controls the format of the file produced.

FRAME/**FORMAT=HDF** appends an HDF raster 8 drawn to the specified or implied input file. The default format is HDF.

FRAME/**FORMAT=GIF** creates a new GIF file, any existing GIF file with the specified or implied name using relative version number or less. Note that in this mode of grabbing an image Ferret creates a GIF by requesting the image back from your screen (your X server). This means that the X server normally has to be configured as pseudo-color. An alternative approach which does not share this restriction is to start Ferret with "ferret -gif" (see p. 6)

FRAME/**FILE=filename** specifies the name of the output file. If /FORMAT is not specified the output format is inferred from filename extensions of .hdf, .HDF, .gif, or .GIF.

The maximum filename length, including path, that is allowable is 255 characters.

## Ref Sec12. GO

/HELP

Executes a list of commands stored in a file.

```
yes? GO file_name
```

If no filename extension is specified a default of .jnl will be assumed. If the full path is specified then the filename must be enclosed in double quotation marks.

The GO command can pass arguments to the script (tool) it executes. See the introductory chapter, section "Writing GO Tools" (p. 19) for more information. Arguments to the GO command may be separated by blanks or commas. To specify multiple words as a single argument, enclose them in quotation marks. To specify an argument that is deliberately omitted, use " " or two consecutive commas.

The response of Ferret to errors encountered during execution of the command file is determined by mode IGNORE_ERRORS. (See command SET MODE, p. 299.)

The echoing of command file lines is controlled by mode VERIFY.

The GO command understands a special syntax called "relative version numbers." If a filename is specified for the GO command which has a version value of zero or less its value is in-

terpreted as relative to the current highest version number. See the chapter "Computing Environment", section "Relative version numbers" (p. 192) for a discussion of relative version numbers of files.

**Note:** The command SET MODE IGNORE_ERRORS is useful when rerunning past sessions which may have errors.

/HELP
The command GO/**HELP filename** opens the named script with the Unix "more" command and displays the first 20 lines of the named file. Use this command to quickly see the documentation in a GO script.

## Ref Sec13. HELP

On Unix systems interactive Ferret help is available from the command line with the commands Fapropos, Fhelp, and Ftoc. If multiple windows are not available on your system the ^Z key can be used to suspend the current Ferret session and access the help; the Unix command "fg" will then restore the suspended session.

See the introductory chapter, section "Unix on-line help" (p. 26) for more information.

## Ref Sec14. IF

Ferret provides an IF-THEN-ELSE syntax to allow conditional execution of commands. It may be used in two styles—single line and multi-line. In both the single and multi-line styles the true or false of the IF condition is determined by case-insensitive recognition of one of these options:

TRUE condition:

- a valid, non-zero numerical value
- TRUE
- T
- YES
- Y

FALSE condition:

- a zero value
- an invalid embedded expression (see next paragraph)
- FALSE
- F
- NO

- N
- BAD
- MISSING

**Examples:**

- `IF `i GT 5` THEN SAY "I is too big" ENDIF`
  writes message if the value of I is greater than 5
- `IF ($yes_or_no) THEN GO yes_script ELSE GO no_script`
  executes yes_script or no_script according to the value of the symbol yes_or_no
- `IF ($dset%|coads>TRUE|%) THEN GO my_plot`
  executes the script my_plot.jnl only if the symbol dset contains "coads"
- ```
  IF `i LT 3` THEN
      GO option_1
  ELIF `i LT 6` THEN
      GO option_2
  ELSE
      GO option_3
  ENDIF
  ```
  uses the multi-line IF syntax to select among GO scripts.

Embedded (grave accent) expressions can be used in conjunction with the IF syntax. For example, `3 GT 2` (Is three greater than 2?) evaluates to "1" (TRUE) and `3 LT 2` (Is three less than 2?) evaluates to "0" (FALSE). If the result of a grave accent expression is invalid, for example division by zero as in `1/0`, the string "bad" is, by default, generated. Thus invalid expressions are regarded as FALSE.

Symbol substitution permits IF decisions to be based on text-based conditions. Suppose, for example, the symbol ($DSET) contains either coads or levitus. Then an IF condition could test for coads using ($DSET%|coads>TRUE|%*>FALSE%).

```
IF ($DSET%|coads>TRUE|*>FALSE%) THEN
      GO cscript
ELSE
      GO lscript
ENDIF
```

The single line style allows IF-THEN-ELSE logic to be applied on a single line. For example, to make a plot only when the surface (Z=0) temperature exceeds 22 degrees we might use

```
IF `TEMP[X=160W,Y=2N,Z=0] GT 22` THEN PLOT TEMP[X=160W,Y=2N]
```

The single line syntax may be any of the following:

IF condition THEN clause_1
IF condition THEN clause_1 ENDIF
IF condition THEN clause_1 ELSE clause_2
IF condition THEN clause_1 ELSE clause_2 ENDIF
```

Note that both ELSE and ENDIF are optional in the single line syntax. Groups of commands enclosed in parentheses and separated by semicolons can be used as clause_1 or as clause_2. There is no ELIF (pronounced "else if") statement in the single line syntax. However, IF conditions can be nested as in

```
IF `i1 GT 5` THEN (IF `j1 LT 4` THEN go option_1 ELSE go option_2)
```

The multi-line style expands the IF capabilities by adding the ELIF statement. Multi-line IF statement follows the pattern

```
IF condition_1 THEN
      clause_1_line_1
      clause_1_line_2
      ...
ELIF condition_2 THEN
      clause_2_line_1
      ...
ELIF condition_3 THEN
      ...
ELSE
      ...
ENDIF
```

Note that THEN is optional at the end of IF and ELIF statements but the ENDIF statement is required to close the entire IF block. Single line IF statements may be included inside of multi-line IF blocks.

Also note that this usage is different from the "masking" IF-THEN-ELSE logic; they share keywords but have different usage. See p. 91

## Ref Sec15. LABEL

/NOUSER

Places a label on the current plot; alias for PPL %LABEL. %LABEL is one of PPLUS's primitive plot commands. It places a label on the plot immediately after being issued (rather than deferring placement). PPLUS does not assign numbers to labels created with LABEL, so they cannot be manipulated as movable labels. The label can also be placed on the plot using the mouse to point and click (see the chapter "Customizing Plots", section "Positioning labels using the mouse pointer," p. 141).

```
yes? LABEL xpos, ypos, center, angle, size   text
```

| | |
|---|---|
| xpos, ypos | position in user units (world coordinates) |
| center | -1 left justification |
| | 0 centered |
| | 1 right justification |
| angle | angle in degrees, 0 degrees at 3 o'clock |
| size | size of text in inches |

See the chapter "Customizing Plots", section "Labels" (p. 136) for examples.

**Command qualifiers for LABEL:**

LABEL/**NOUSER**
Locates labels in inches instead of user units (xpos and ypos are specified in inches rather than in world coordinates).

## Ref Sec16. LET

The LET command is an alias for DEFINE VARIABLE (p.260 ). All qualifiers and restrictions are identical to DEFINE VARIABLE.

**Example:**

```
yes? LET A = B
```

is equivalent to

```
yes? DEFINE VARIABLE A = B
```

## Ref Sec17. LIST

/I/J/K/L /X/Y/Z/T /D /ILIMITS /JLIMITS /KLIMITS /LLIMITS /XLIMITS /YLIMITS /ZLIMITS /TLIMIT /APPEND /FILE /FORMAT /HEADING /NOHEAD /TITLE /ORDER /RIGID /PRECISION /CLOBBER /SINGLE /QUIET

Produces a listing of the indicated data.

```
LIST[/qualifiers]   [expression_1 , expression_2 , ...]
```

**Example:**

```
yes? LIST/Z=10 u , v , u^2 + v^2
```

Lists the 3 quantities specified using the current default data set and region (at depth 10).

**Parameters**

Expressions may be any valid expression. See the chapter "Variables and Expressions", section "Expressions" (p. 53) for a definition of valid expressions. If multiple variables or expressions are specified they may be listed together in columns or in sequence depending on the /SINGLY qualifier. The expression(s) will be inferred from the current context if omitted from the command line.

If multiple expressions are given on the command line and /SINGLY is not specified, then the expressions must be conformable. See the chapter "Variables and Expressions", section "Multi-dimensional expressions" (p. 54) for a definition of conformable expressions. Degenerate or single point axis limits will be promoted up (values repeated) as needed.

**Example:**

```
yes? LIST/I=1:3/J=1:2  i+j, i
```

**Command qualifiers for LIST:**

LIST/**I= /J= /K= /L=/X= /Y= /Z= /T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression(s) being listed.

LIST/**ILIMITS=/JLIMITS=/KLIMITS=/LLIMITS=**
Specifies the size of the desired NetCDF output file independently from the actual data being saved. By specifying axis limits in excess of the saved expression's limits it is possible to /APPEND data later. (See the chapter "Converting to NetCDF", section "Simple Conversions Using Ferret," p. 193, ex. 4).

LIST/**XLIMITS=/YLIMITS=/ZLIMITS=/TLIMITS=**
Specifies the size of the desired NetCDF output file independently from the actual data being saved. By specifying axis limits in excess of the saved expression's limits it is possible to /APPEND data later. (See the chapter "Converting to NetCDF", section "Simple Conversions Using Ferret," p. 193, ex. 4).

LIST/**D=**
Specifies the default data set to be used when evaluating the expression(s) being listed.

LIST/**APPEND**
Use this qualifier together with the /FILE qualifier to indicate that the listed data should be appended to a pre-existing file. If no file exists by the name indicated a new file is created. This qualifier is not applicable to /FORMAT=GT. When used with /FORMAT=CDF it permits any data in the file to be overwritten, new variables to be added to the file, and appending of new indices along the T axis of the variables in the file. This qualifier overrides the command CANCEL LIST/APPEND.

LIST/**FILE**[=file_name]
Names a file to receive the listed data. If /FILE is specified with no name then the default name is used from the SET LIST/FILE command.

**Example:**

```
yes? LIST/FILE=my_file.dat sst[D=coads_climatology]
```

See command SET LIST (p. 296) for further information on automatic filename generation.

LIST/**CLOBBER**
Used with LIST/FILE. Indicates that any existing file with the name used is to be deleted, before writing. If CLOBBER is not specified and the file exists, and error message is given.

**Example:**

```
yes? LIST/FILE=my_file.dat/CLOBBER sst[D=coads_climatology]
```

LIST/**FORMAT=**
Specifies an output format ( =format_choice) for the data to be listed.

```
yes? SET LIST/FORMAT=format_choice
                or
yes? SET LIST/FORMAT     (use format set by SET LIST/FORMAT)
```

Format choices:

| | |
|---|---|
| FORTRAN format | produces ASCII output |
| "UNFORMATTED" | produces unformatted (binary) output using FORTRAN record structure |
| "CDF" | produces NetCDF format output |
| "GT" | produces TMAP GT format |
| "STREAM" | produces unstructured binary floating point (C-style) |
| "tab" | produces tab-delimited output |
| "comma" | produces comma-delimited output |

This command has the same function as SET LIST/FORMAT except that it does not affect future LIST commands. See command SET LIST/FORMAT (p. 297) for detailed documentation.

**Notes for LIST/FORMAT:**

1) All output values, regardless of the /FORMAT designation, will be of type single precision floating point. For FORTRAN output formats this means all numerical field specifiers must be "F", "E", or "G".

2) For FORTRAN-formatted and UNFORMATTED (binary) output, the contents of a single output "record" are determined by the /ORDER qualifier. For example, each record will be a line of Y values for LIST/ORDER=YX. If /ORDER is omitted, the records will be the first output axis of greater than unity length taken in the order X, Y, Z, then T. FORTRAN-formatted output records may be further split by the usual rules of FORTRAN output formatting.

3) FORTRAN formats must be enclosed in parentheses. If blanks are included in the format it must be enclosed in quotation marks. Output strings are permitted in the format.

   **Example:**

```
yes? LIST/FORMAT=("The temperature is:", F6.3)  sst[X=180, Y=0]
```

4) When FORTRAN formats are used, and more than one value per record is desired, the /ORDER qualifier ( p274) must be used, even if the variable is defined along only one axis.

   **Example:**

```
yes? LIST/FORMAT=(8F6.3)/ORDER=T  sst[X=180, Y=0]
```

5) The default listing style includes labels for the rows and columns of the output. When a FORTRAN format is specified, these labels are omitted.

6) On Unix systems the /FORMAT=UNFORMATTED specifier produces FORTRAN-style variable-length records. On most implementations this means that a 4-byte field containing the record length begins and ends each record of data.

7) The command alias SAVE is provided for the commonly used LIST/FORMAT=CDF. NetCDF outputs are self-documenting, including grid definitions. The output files can be used as input with the command USE—alias for SET DATA/FORMAT=CDF. See command SAVE (p. 287) for further notes about NetCDF files.

LIST/**HEAD**
For ASCII data listings this command determines whether to precede the listing with a heading describing data set, variable and region. This qualifier overrides the CANCEL LIST/HEAD command.

LIST/**HEADING[=ENHANCED]**
For ASCII data listings this qualifier determines whether to precede the listing with a heading that describes the data set, variable, and region. This qualifier overrides the CANCEL LIST/HEAD command. When the argument /HEADING=ENHANCED is used a self-documenting heading is provided that includes the axis coordinates.

For NetCDF output files (alias SAVE) the /HEADING=ENHANCED option causes the NetCDF file structure to include extra coordinate information that describes how the particular

data subset being written fits within the broader coordinate system of the grid from which it is extracted. When a NetCDF file with an enhanced heading is accessed by Ferret (using SET DATA or USE) the index values will appear to be consistent with the parent data set.

LIST/**NOHEAD**
Does not precede listing with a heading describing data set, variable and region. This qualifier overrides the SET LIST/HEAD command.

LIST/**ORDER=**
Specifies the order (ORDER=permutation) in which axes are to be laid out in the listing.

**Examples:**

```
yes? LIST/ORDER=XY sst          !X varies fastest
yes? LIST/ORDER=YX sst          !Y varies fastest
```

The "permutation" string may be any permutation of the letters X, Y, Z, and T. /ORDER is applicable only to /FORMAT=unf and FORTRAN formats.

Note that a 1-dimensional list will, by default, place only one value per record. The /ORDER qualifier can cause the 1-dimensional list to occur in a single record. For example,

```
LIST/I=1:5 I
```

will list as 5 records whereas

```
LIST/I=1:5 /ORDER=X I
```

will list 5 values on a single record.

LIST/**PRECISION**=#
Controls the digit precision of LIST output

Using the qualifier /PRECISION=#digits the output precision of the LIST command may be easily controlled. This qualifier functions exactly as does the SET LIST/PRECISION= command but it applies only to the current command.

LIST/**QUIET**
Using the qualifier /QUIET will prevent the message "LISTing to file XXXX.XXXX" from being displayed.

LIST/**RIGID**
Valid only with /FORMAT=CDF. Indicates that Ferret should not create a NetCDF "record" axis as the time axis for any of the variables listed with this command. Time axes are, instead, of fixed length and the /APPEND qualifier is not usable to extend the listing.

LIST/**SINGLY**

This qualifier is relevant only when multiple expressions are specified in the LIST command. When the /SINGLY qualifier is specified the entire listing of each expression including (optional) heading and all data is completed before proceeding to the next expression.

By default the expressions are not listed singly—each line contains one value of each expression. The qualifier has no effect if only a single expression is specified. If the /FILE qualifier is specified to use automatic filename generation and /APPEND is not specified, then each expression is listed to a separate file.

LIST/**TITLE="title string"**
Valid only with /FORMAT=CDF. Causes the global attribute "title" to be defined in a NetCDF file, thereby setting its title.

## Ref Sec18. LOAD

/I/J/K/L /X/Y/Z/T /D /NAME /PERMANENT /TEMPORARY

Loads a variable or expression into memory.

```
yes? LOAD[/qualifiers]   [expression_1 , expression_2 , ...]
```

Loading may speed execution of later commands that will require the loaded data. Often it is helpful to LOAD a large region of data encompassing several small regions in which the analysis will be pursued.

Load interacts with the current context exactly as other "action" commands CONTOUR, PLOT, SHADE, VECTOR, LIST, etc. do.

**Parameters**

Expressions may be any valid expression. See the chapter "Variables and Expressions", section "Expressions" (p. 53) for a definition of valid expressions. If multiple variables or expressions are specified they are treated in sequence. The expression(s) will be inferred from the current context if omitted from the command line.

**Command qualifiers for LOAD:**

LOAD/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression(s) being loaded.

LOAD/**D=**
Specifies the default data set to be used when evaluating the expression(s) being loaded.

LOAD/**NAME**

Obsolete. Provided for compatibility with much older Ferret versions.

LOAD/**PERMANENT**
Data loaded with LOAD/PERMANENT are kept in memory until a LOAD/TEMPORARY command is given that refers to the same data. See command LOAD/TEMPORARY (p. 276). Note that this command may cause memory fragmentation. It should generally be given immediately following CANCEL MEMORY and preferably is used only to load file variables (as opposed to expressions).

LOAD/**TEMPORARY** (default)
Data loaded with LOAD or LOAD/TEMPORARY is brought into memory but may be unloaded based on a priority scheme of least recent use when memory space is required.

## Ref Sec19. MESSAGE

/CONTINUE /QUIET /JOURNAL /ERROR

Displays a message at the terminal.

```
yes? MESSAGE text
```

By default a carriage return is required from the keyboard for program execution to continue (used to halt the execution of a command file).

**Command qualifiers for MESSAGE:**

MESSAGE/**CONTINUE**
Continues program execution following the display of the message text without waiting for a carriage return from the operator.

MESSAGE/**JOURNAL**
Writes the message to the journal file.

MESSAGE/**ERROR**
Writes the message to standard error.

MESSAGE/**QUIET**
Waits for a carriage return from the operator but does not supply a prompt for it.

## Ref Sec20. PALETTE

Alias for PPL SHASET SPECTRUM=. Specifies or restores the default color.

```
yes? PALETTE pal_name
```

The argument is the name of a palette file. Many palettes are included in the Ferret distribution. Try the Unix command "Fpalette '*'" to see a list of available palette files.

Some of the palettes are designed for particular needs. "centered.spk", for example, emphasizes the contrast between positive and negative shade levels. "land_sea.spk" uses blue tones for negative values and browns and greens for positive values, making it suitable for topography displays.

Palette files end in the file suffix .spk, but the suffix is not necessary when specifying a palette. Use `GO try_palette pal_name` to display a palette. The GO files "exact_color.jnl" and "squeeze_colors.jnl" can be used to modify palettes. You can also create new palette files with a text editor. See the chapter "Customizing Plots", section "Shade and fill colors" (p. 145) for the format of a palette file.

PALETTE with no argument restores the default palette. When you use the qualifier /PALETTE= in conjunction with /SET_UP, PPLUS makes the specified color spectrum the new default palette, and all subsequent shaded or color-filled plots will use that palette as the default. To restore the previous palette to the default, use PALETTE with no argument after your customization.

## Ref Sec21. PATTERN

Alias for PPL PATSET PATTERN=. Specifies or restores the default pattern.

```
yes? PATTERN patt_name
```

The argument is the name of a pattern file. Many patterns are included in the Ferret distribution. Try the Unix command "Fpattern '*'" to see a list of available pattern files.

Ferret has the capability to make color fill plots using solid color only, and also with colors laid on in patterns.

The PATTERN command sets the patterns to be used in a plot generated with the SHADE, FILL and POLYGON commands. It is similar to the PALETTE command, which sets colors, but the PALETTE and PATTERN commands act independently.

When Ferret is started up, only one pattern is set, SOLID. The SOLID pattern is equivalent to not using any pattern, and SHADE, FILL and POLYGON fill their cells with solid color.

Pattern files end in the file suffix .pat, but use of the suffix is not necessary when specifying a pattern. Use `GO try_pattern patt_name` to display the patterns specified in a pattern file. `GO show_all_patterns` draws a plot showing all the available pattern files and their names. No-

tice that patterns can be used with a single color, or multiple colors, depending entirely on the PALETTE specification.

A pattern file may specify one or more patterns. If there are fewer patterns specified in a pattern file than there are levels in a particular plot, the patterns will be repeated.

## Ref Sec22. PAUSE

Alias for MESSAGE

## Ref Sec23. PLOT

/I/J/K/L /X/Y/Z/T /D /FRAME /LINE /NOLABEL /OVERLAY/SET_UP /SYMBOL /TITLE /TRANSPOSE /VS /HLIMITS /VLIMITS /XLIMITS /YLIMITS /COLOR /SIZE /THICKNESS

Produces a line plot.

```
yes? PLOT[/qualifiers]   [expression_1 , expression_2 , ...]
```

The indicated expression(s) must represent a line (not a plane) of data (PLOT/VS is an exception). Unless the /VS qualifier is used, the independent variable is the underlying coordinate axis for this line of data.

**Example:**

```
yes? PLOT/l=1:100 sst
```

produces a time series plot of the first 100 points of sst.

**Parameters**

The argument(s) for PLOT specify the variable or expression to be plotted.

When the /VS qualifier is used the indicated expressions may have any geometry in 4D space but they must match in the total number of points in each expression. The points are associated in the order of their underlying axes. When the /VS qualifier is not used the indicated expression(s) must describe a line (not a plane) of data.

The expression(s) are inferred from the current context if omitted from the command line—i.e., if no expression is given then the argument most recently given is used, or the default expression may be explicitly set with SET EXPRESSION.

When Ferret plots multiple data lines simultaneously, PPLUS automatically cycles through pen colors and symbols, creating up to 26 distinct line types. Try GO line_samples to see samples of these styles.

**Command qualifiers for PLOT:**

PLOT/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression(s) being plotted.

PLOT/**D=**
Specifies the default data set to be used when evaluating the expression(s) being plotted.

PLOT/**FRAME**
Causes the graphic image produced to be captured as an animation frame and written to the movie file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

PLOT/**COLOR=/THICKNESS=**
Simple syntax for line plots. For line plots it is possible with these qualifiers to control line thickness and color with commands such as

```
yes?  PLOT/COLOR=blue/THICK=2 I[i=1:3]
```

This is equivalent to the (still supported) use of the /LINE qualifiers in

```
yes? PLOT/LINE=10 I[i=1:3] ! 4(blue) + 6*(2-1)
```

The available color names are Black, Red, Green, Blue, LightBlue, Purple, and White (not case sensitive), corresponding to the /LINE values 1-6, respectively. (/COLOR also accepts numerical values.) The line thickness may be 1, 2, or 3 corresponding to pixel thickness on the screen or corresponding to multiples of the default line thickness on hard copy. Note that White is only available for THICKNESS=1 (the default thickness).

PLOT/**COLOR=/SIZE=/THICKNESS**
Simple syntax for plots using symbols. For symbol (scatter) plots (PLOT/VS or PLOT/SYMBOL), control the color, size, and line thickness of the symbols with commands such as:

```
yes?  PLOT/COLOR=red/SIZE=0.2/THICKNESS=2/SYMBOL=4 I[i=1:5]
```

The available color names are Black, Red, Green, Blue, LightBlue, Purple, and White (not case sensitive), corresponding to the /LINE values 1-6, respectively. (/COLOR also accepts numerical values.) The line thickness may be 1, 2, or 3 corresponding to pixel thickness on the screen or corresponding to multiples of the default line thickness on hard copy; note that White is only available in the default THICKNESS=1. The /SIZE is given in units of "inches", consistent with the PLOT+ usage of "inches". (These are the same units as in, say, "ppl axlen 8,6", to specify plot axes of lengths 8 and 6 inches for horizontal and vertical axes, respectively.)

PLOT/**LINE[=]**
The /LINE qualifier without =n causes the PLOT command to connect the plotted points with a line regardless of the state of the /SYMBOLS qualifier.

For simpler specification of line characteristics see PLOT/COLOR=/THICKNESS= above ( p. 279) . /LINE=n specifies the line style. "n" is an integer between 1 and 18. `GO line_thickness` draws samples of the available line styles. Line style "1" is always a solid line in the foreground color (black or white). Other line styles are device dependent (colors or dash patterns). For color devices, n=1–6 draws single-thickness lines each a different color. n=7–12 draws double-thick lines in the same color order, and n=13–18 draws triple-thick lines. See the chapter "Customizing Plots", section "Text and line colors" (p. 143) for a chart of the default colors.

PLOT/**NOLABELS**
Suppresses all plot labels except axis labels.

PLOT/**OVERLAY**
Causes the indicated field(s) to be overlaid on the existing plot. This qualifier can also be used to overlay lines or symbols on 2D plots (SHADE, CONTOUR, or VECTOR) provided the axis scalings are appropriate.

PLOT/**SET_UP**
Performs all the internal preparations required by program Ferret for plotting but does not actually render the plot. The command PPL can then be used to make changes to the plot prior to producing output with the PPL PLOT command. This makes possible certain customizations that are not possible with Ferret command qualifiers. See the chapter "Customizing Plots" (p. 129).

PLOT/**SYMBOL[=]**
The /SYMBOL qualifier causes the PLOT command to mark each plotted point with a symbol. If the /LINE qualifier is given too the symbols are also connected with a line; if /LINE is omitted no connecting line is drawn.

Optionally, the symbol number may be explicitly specified as an integer value between 1 and 88. The integer refers to the PPLUS plot marker numbers (e.g., 1 for x, 3 for +, etc.). Type "GO show_symbols" and "GO show_88_syms" at the Ferret prompt to see available symbols and their reference numbers. The symbols are also documented on page 1 of the document $FER_DIR/doc/pplus_fonts.ps.

PLOT/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about the expression(s). To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character.

PLOT/**TRANSPOSE**

Causes the horizontal and vertical axes to be interchanged. By default the X axis is drawn horizontally on the plot and the Y and Z axes are drawn vertically. For Y-Z plots the Z data axis is vertical by default.

PLOT/**VS**
Specifies that the first expression given in the command line is to be used as the independent axis.

**Example:**

```
yes? PLOT/Y=20S:20N/X=180/T=27740:27741/Z=100/VS  temp , salt
```

Produces a plot of salinity (vertical axis) against temperature (horizontal axis) along the indicated range of latitudes and times. The plot will be labeled "salt"; the vertical (dependent) variable is the one that determines the key. The qualifier /TRANSPOSE can be used in conjunction with /VS to further manipulate the labeling and axis orientation.

PLOT/VS implies /SYMBOL by default to produce scatter plots. Use PLOT/VS/LINE to produce a line plot.

PLOT/**HLIMITS=**
Specifies axis range and tic interval for the horizontal axis. Without this qualifier Ferret selects a reasonable range.

```
yes? PLOT/HLIMITS=lo:hi:[increment]  [expression(s)]
```

The optional "increment" parameter determines tic mark spacing on the axis. If the increment is negative, the axis is reversed.

The /HLIMITS and /VLIMITS qualifiers will retain their "horizontal" and "vertical" interpretations in the presence of the /TRANSPOSE qualifier. Thus, the addition of /TRANSPOSE to a plotting command mandates the interchange of "H" and "V" on the limits qualifiers.

PLOT/**VLIMITS=**
Specifies the axis range and tic interval for the vertical axis. See /HLIMITS (above).

PLOT/**XLIMITS=**
**Note: XLIMITS** and **YLIMITS** have been denigrated. Please use **HLIMITS** and **VLIMITS** instead.
Specifies axis range and tic interval for the X axis. Without this qualifier Ferret selects a reasonable range.

```
yes? PLOT/XLIMITS=lo:hi:[increment]  [expression(s)]
```

The optional "increment" parameter determines tic mark spacing on the axis. If the increment is negative, the axis is reversed.

Note that the “X” in /XLIMITS refers to the horizontal axis of the plot rather than to the X axis of the grid. Plots may be transposed manually with the /TRANSPOSE qualifier. On transposed plots /XLIMITS will refer to the vertical axis of the plot.

PLOT/**YLIMITS=**
**Note: XLIMITS** and **YLIMITS** have been denigrated. Please use **HLIMITS** and **VLIMITS** instead.
Specifies the axis range and tic interval for the Y axis. See /XLIMITS (above).

## Ref Sec24. POLYGON

/I/J/K/L/X/Y/Z/T/OVERLAY/SET_UP/FRAME/D /TRANSPOSE /COORD_AX /SYMBOL /NOLABELS /LEVELS /LINE /COLOR /PALETTE /TITLE /THICKNESS /XLIMITS /YLIMITS /NOAXES /PATTERN /FILL /KEY /NOKEY/HLIMITS /VLIMITS

Produces a color-filled or line plot of polygons. By default a color key is drawn and lines are not drawn.

```
POLYGON[/qualifiers]  x-vertices, y-vertices [, values]
```

**Parameters**

The two x- and y- vertices parameters separately specify the x and y coordinates of the vertices of the polygons to be plotted.

The values may be any valid expression. If a color-filled plot is specified, the numerical value of the expression associated with each polygon determines the color of that polygon, as in SHADE and FILL plots. See the chapter “Variables and Expressions”, section “Expressions” (p. 53) for a definition of valid expressions. If values are omitted the /FILL option is not valid—only /LINE plots may be made.

**Example:**

```
yes? LET XTRIANGLE = YSEQUENCE({-1,0,1})
yes? LET YTRIANGLE = YSEQUENCE({-1,1,-1})
yes? LET XPTS = 180+30*RANDU(I[i=1:10])
yes? LET YPTS = 30*RANDU(1+I[i=1:10])
yes? POLYGON XTRIANGLE+XPTS, YTRIANGLE+YPTS, I[I=1:10]
```

**Command qualifiers for POLYGON:**

POLYGON /**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression being plotted.

POLYGON/**D=**

Specifies the default data set to be used when evaluating the expression being plotted.

POLYGON/**FRAME**
Causes the graphic image produced by the command to be captured as an animation frame in the file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

POLYGON/**KEY**
Displays a color key for the palette used in the color-filled plot. By default a key is drawn unless the /LINE or /NOKEY qualifier is specified.

POLYGON/**LEVELS**
Specifies the POLYGON levels or how the levels will be determined. If the /LEVELS qualifier is omitted Ferret automatically selects reasonable POLYGON levels.

See the chapter "Customizing Plots", section “Contouring” (p. 154) for examples and more documentation on /LEVELS.

POLYGON/**LINE**
Outlines polygons specified by x and y vertices on a POLYGON plot. When /LINE is specified the color key is omitted unless specifically requested via /KEY.

POLYGON/**LINE/COLOR=/THICK=**
Simple specification of outline characteristics f or polygon plots which specify an outline line we control line thickness and color with commands such as

```
yes?  POLYGON/LINE/COLOR=blue/THICK=2  {1,2,1}, {3,2,1}
```

This is equivalent to the (still supported) use of the /LINE qualifiers in

```
yes?  POLYGON/LINE=10 {1,2,1}, {3,2,1}  ! 4(blue) + 6*(2-1)
```

The available color names are Black, Red, Green, Blue, LightBlue, Purple, and White (not case sensitive), corresponding to the /LINE values 1-6, respectively. (/COLOR also accepts numerical values.) The line thickness may be 1, 2, or 3 corresponding to pixel thickness on the screen or corresponding to multiples of the default line thickness on hard copy, however the color White is only available in the default THICKNESS=1.

**COLOR**
Specifies the POLYGON levels or how the levels will be determined. If the /LEVELS qualifier is omitted Ferret automatically selects reasonable P

POLYGON/**NOKEY**
Suppresses the drawing of a color key for the palette used in the plot.

POLYGON/**NOLABELS**

Suppresses all plot labels except axis labels.

POLYGON/**OVERLAY**
Causes the indicated POLYGON plot to be overlaid on the existing plot.

POLYGON/**PALETTE=**
Specifies a color palette (otherwise, a default rainbow palette is used). Try the Unix command `% Fpalette '*'` to see available palettes. The file suffix *.spk is not necessary when specifying a palette. See command PALETTE (p. 276) for more information.

The /PALETTE qualifier changes the current palette for the duration of the plotting command and then restores the previous palette. This behavior is not immediately compatible with the /SET_UP qualifier. See the PALETTE command (p. 276) for further discussion.

POLYGON/**PATTERN=**
Specifies a pattern file (otherwise, a default SOLID pattern is used). Try the Unix command `% Fpattern '*'` to see available pattern files. The file suffix *.pat is not necessary when specifying a pattern file. See command PATTERN (p. 277) for more information.

POLYGON/**SET_UP**
Performs all the internal preparations required by program Ferret for a POLYGON plot but does not actually render output. The command PPL can then be used to make changes to the plot prior to producing output with the PPL FILLPOL command. This permits plot customizations that are not possible with Ferret command qualifiers. See the chapter "Customizing Plots" (p. 129).

POLYGON/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about the expression(s). To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character. See the chapter "Customizing Plots", section “Fonts” (p. 149).

```
yes? POLYGON/TITLE="title string" x-vertices, y-vertices, values
```

POLYGON/**TRANSPOSE**
Causes the horizontal and vertical axes to be interchanged. By default the X axis is drawn horizontally on the plot and the Y and Z axes are drawn vertically. For Y-Z plots the Z data axis is vertical.

Note that plots in the YT and ZT planes have /TRANSFORM applied by default in order to achieve a horizontal T axis. See /XLIMITS (below) for further details. Use /TRANSPOSE manually to reverse this effect.

POLYGON/**HLIMITS=**

Specifies the horizontal  axis range and tic interval (otherwise, Ferret selects reasonable values).

```
yes? POLYGON/HLIMITS=lo:hi:increment
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

The /HLIMITS and /VLIMITS qualifiers will retain their "horizontal" and "vertical" interpretations in the presence of the /TRANSPOSE qualifier.  Thus, the addition of /TRANSPOSE to a plotting command mandates the interchange of "H" and "V" on the limits qualifiers.

POLYGON/**VLIMITS=**

Specifies the vertical axis range and tic interval. See /HLIMITS (above)

POLYGON/**XLIMITS=**
**Note: XLIMITS** and **YLIMITS** have been denigrated.  Please use **HLIMITS** and **VLIMITS** instead.
Specifies the X axis range and tic interval (otherwise, Ferret selects reasonable values).

```
yes? POLYGON/XLIMITS=lo:hi:increment
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

Note that the “X” in /XLIMITS refers to the horizontal axis of the plot rather than to the X axis of the grid. This can lead to confusion, especially on plots in the YT or ZT plane. Plots in these planes are automatically transposed to place the Y or Z axis, respectively, on the vertical axis of the plot. Plots may also be transposed manually with the /TRANSPOSE qualifier. On transposed plots /XLIMITS will refer to the vertical axis of the plot.

POLYGON/**YLIMITS=**
Note: XLIMITS and **YLIMITS** have been denigrated.  Please use **HLIMITS** and **VLIMITS** instead.
Specifies the Y axis range and tic interval. See /XLIMITS (above).

## Ref Sec25.  PPLUS

/RESET

Invokes PPLUS (“PLOT PLUS” written by Don Denbo),  to execute a command or commands.

```
yes? PPLUS                !(also PPL); invokes PPLUS interactively
```
or
```
yes? PPL pplus_command    !executes a single PPLUS command
```
or
```
yes? PPL/RESET            !restores PPLUS to start-up defaults
```

**Example:**

```
yes? PPL CROSS 1            !reference line through zero
```

Executes the PPLUS command “CROSS” and immediately returns control to Ferret.

When PPLUS is invoked interactively the prompt is “PPL>” instead of the usual “yes?”. The EXIT command given at the “PPL>” prompt returns control to Ferret.

See the chapter "Customizing Plots" (p. 129) for more information on Ferret/PPLUS interactions. A complete list of PPLUS commands is in PLOT PLUS for Ferret User’s Guide.

**Command Qualifiers for PPLUS:**

PPLUS/**RESET**
Restores PPLUS to start-up settings.

## Ref Sec26. QUIT

Alias for EXIT; also just Q.

## Ref Sec27. REPEAT

/I/J/K/L /X/Y/Z/T

Repeats a command or group of commands over a range of values along an axis.

```
yes? REPEAT/q=lo:hi[:increment] COMMAND
```

The units of lo, hi, and increment are the units of the underlying grid axis if the qualifier is X, Y, Z, or T. The qualifiers I, J, K, or L advance the repeat loop by incrementing the indicated index (the default index increment is 1). Use SHOW GRID to examine the axis units (if the units are not displayed try CANCEL MODE LATITUDE, LONGITUDE, or CALENDAR as appropriate). To run the loop from the highest value decreasing towards the lowest value, specify increment to be less than zero. Any command or group of commands that can be specified at the command line can also be given as an argument to REPEAT. If MODE VERIFY is SET, the current loop index is displayed at the console as REPEAT executes.

**Examples:**

1) `yes? REPEAT/L=1:240 CONTOUR/Y=30S:50N/X=130E:70W/LEV/FRAME sst`
   Produces a 240-frame movie of sea surface temperature.

2) `yes? REPEAT/Z=300:0:-30 GO compz`
   Executes the command file compz.jnl at Z=300, Z=270, ..., Z=0.

3) `yes? REPEAT/L=1:250:5 (GO set_up; CONTOUR sst; FRAME)`
   Repeats three commands—execution of a GO script, CONTOUR, and FRAME—for each timestep specified.

**Command qualifiers for REPEAT:**

REPEAT/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Repeats the requested command(s) for the specified range of axis subscripts (I, J, K, or L) or axis coordinates (X, Y, Z, or T). Note that when T axis limits are specified as dates, the units of increment are hours.

## Ref Sec28. SAVE

The SAVE command is an alias for LIST/FORMAT=CDF (p.272 ). All qualifiers and restrictions are identical to LIST/FORMAT=CDF.

**Example:**

```
yes? SAVE temp, salt
```
is identical to
```
yes? LIST/FORMAT=CDF temp, salt
```

**Notes:**

1) Gaps in NetCDF outputs are filled with the missing value flag of the variable being written. (See the chapter "Variables and Expressions", section “Missing value flags,” p. 51.) In the example below, if “temp” and “salt” share the same time axis then the L=2:4 values of salt will be so filled.

   ```
   yes? SAVE/FILE=test.cdf   temp[L=1:5], salt[L=1], salt[L=5]
   ```

2) Transformations that compress an axis to a point produce results that Ferret regards as time-independent. Thus, this 12-month average:

   ```
   yes? SAVE/FILE=annual.cdf  sst[L=1:12@AVE]
   ```

creates a NetCDF file with no time axis. It would not be possible to append the average of the next 12 months as the next time step of this file. However, a time location can be inherited from another variable. In this example, we inherit the time axis of "timestamp" in order to create a time axis in the NetCDF file.

```
yes? DEFINE AXIS/T="1-JUL-1980":"1-JUL-1985"/UNIT=year  tannual
yes? DEFINE GRID/T=tannual  gannual
yes? LET  timestamp = T[G=gannual] * 0              !always 0
yes? LET  sst_ave = sst[L=1:12@AVE] + timestamp
yes? SAVE/FILE=annual.cdf  sst_ave[L=1]
yes? LET  sst_ave = sst[L=13:24@AVE] + timestamp
yes? SAVE/FILE=annual.cdf/APPEND  sst_ave[L=2]
.
.
. etc.
```

3) Background documentation about the definition and data set of origin for a variable are saved in the "history" attribute of a variable when it is first saved in the NetCDF file. If the definition of the variable is then changed, and more values are inserted into the file using SAVE/APPEND, the modified definition will NOT be documented in the output file. If the new definition changes the defining grid for the variable the results will be unpredictable.

## Ref Sec29. SET

Sets features of the operating environment for program Ferret.

Generally, features may be toggled on and off with SET and CANCEL. Features affected by SET may be examined with SHOW (see also CANCEL, p. 241, and SHOW, p. 317).

## Ref Sec29.1. SET AXIS

/MODULO

Indicates that an axis is to be treated as a modulo axis (the first point "wraps" and follows the last point, as in a longitude axis).

```
yes? SET AXIS/MODULO  x_ax
```

/DEPTH

Indicates that an axis is to be treated as a depth axis (graphics made with positive down).

```
yes? SET AXIS/DEPTH z_ax
```

## Ref Sec29.2. SET DATA_SET

/EZ /VARIABLE /TITLE /FORMAT /GRID /SKIP /COLUMNS /SAVE /RESTORE /ORDER /TYPE /SWAP /REGULART

**SET DATA/EZ** /COLUMNS /FORMAT /GRID /SKIP /TITLE /VARIABLE

Specifies ASCII, binary, NetCDF, GT, or TS-formatted data set(s) to be analyzed.

1) ASCII or binary:

```
yes? SET DATA/EZ[/qualifiers] data_set1, data_set2, …
```

or equivalently, with alias FILE:

```
yes? FILE[/qualifiers] data_set1, data_set2, ...
```

2) NetCDF:

```
yes? SET DATA/FORMAT=cdf  NetCDF_file
```

or equivalently, with alias USE

```
yes? USE NetCDF_file
```

3) GT or TS-formatted:

```
yes? SET DATA data_set1, data_set2, ...
```

In the case of GT or TS-formatted files, an extension of .des is assumed. A previously SET data set can be SET by its reference number, as shown by SHOW DATA, rather than by name.

If a Unix filename includes a path (with slashes) then the full path plus name must be enclosed in double quotation marks.

```
yes? use "/home/mydirectory/mydata/new_salinity.cdf"
```

If the filename begins with a numeric character, Ferret does not recognize the file, but it can be specified using the Unix pathname, e.g.

```
yes? use "./123"
```

or

```
yes? file/var=a "./45N_180W.dat"
```

**Note:** Maximum simultaneous data sets: 60 (as of Ferret ver. 3.1). Use CANCEL DATA if the limit is reached.

**Command qualifiers for SET DATA_SET:**

SET DATA/**FORMAT=**
Specifies the format of the data set(s) being SET. Allowable values for "file_format" are "cdf", "free", "unformatted", "stream" or a FORTRAN format in quotation marks and parentheses.

```
yes? SET DATA/FORMAT=file_format [data_set_name_or_number]
```

Valid arguments for /FORMAT=

1) free (default for SET DATA/EZ)
   To use the format "free" a file must consist entirely of numerical data separated by commas, blanks or tabs.

2) cdf
   If SET DATA/FORMAT=cdf (alias USE) is used, the data file must be in CDF format. The default filename extension is ".cdf".

   **Example:**

```
yes? SET DATA/FORMAT=CDF  my_netcdf
        or equivalently,
yes? USE  my_netcdf
```

   See the chapter "Data Set Basics", section "NetCDF data, p. 30."

   **Command qualifiers for SET DATA_SET/FORMAT:**

   SET DATA /FORMAT=CDF /ORDER=<permutation> /REGULART

   The permutation argument contains information both about the order of the axes in the file and the direction.

   The order indicated through the /ORDER qualifier should always be exactly the reverse of the order in which the dimensions of variables as revealed by the netCDF ncdump -h command are declared. (This ambiguity reflects the linguistic difference between "C ordering" and "FORTRAN ordering". The default X-Y-Z-T ordering used in the COARDS standard and in Ferret documentation would be referred to as T-Z-Y-X ordering if we used C terminology.)

   Thus, to USE a file in Ferret in which the data on disk transposes the X and Y axes we would specify

   USE/ORDER=YX my_file.nc

To use a file in which the data were laid down in XZ "slabs", such as might occur in model outpus we would specify

```
USE/ORDER=XZYT my_model.nc
```

To indicate that the coordinates along a particular axis are reversed from the "right hand rule" ordering, for example a Y axis which runs north to south (not uncommon in image data), we would precede that axis by a minus sign. For example

```
USE/ORDER=-XY my_flipped_images.nc
```

The minus sign should be applied to the axis position **after** transposition. Thus if a file both transposed the XY axis ordering and used north-to-south ordering in latitude one would access the file with

```
USE/ORDER=Y-X my_transposed_flipped_images.nc
```

NetCDF files, while in principle self-documenting, may be contain axis ambiguities. For example, a file which is supposed to contain a time series, but lacks units on the coordinate variable in the file may appear to be a line of data on the X axis. The /ORDER qualifier can be used to resolve these ambiguities. For this example, one would initialize the file with the command

```
USE/ORDER=T my_ambiguous_time_series.nc
```

**Notes for USE/ORDER:**

1) Note that specifying USE/ORDER=XYZT is not always equivalent to specifying default ordering. For example, if a netCDF file contained variables on an XYT grid, the /ORDER=XYZT specifaction would tell Ferret to interpret it as an XYZ grid.

2) Also note, the /ORDER qualifier will be ignored if either the file is not netCDF, or the file is netCDF but has an "enhanced" header (see SAVE/HEADING=enhanced, p 273)

3) unformatted
To use the format “unformatted” the data must be floating point, binary, FORTRAN-style records with all of the desired data beginning on 4-byte boundaries. This option expects 4 bytes of record length information at the beginning and again at the end of each record. The “-” designator ( /VARIABLES) can be used to skip over unwanted 4-byte quantities (variables) in each record. See the chapter “Data Set Basics”, section “Binary data” (p. 33).

4) FORTRAN format string
FORTRAN format specifications should be surrounded by parentheses and enclosed in quotation marks.

**Example:**

```
yes? SET DATA/EZ/FORMAT="(5X,F12.0)" my_data_set
```

or equivalently,

```
yes? FILE/FORMAT="(5X,F12.0)" my_data_set
```

5) stream (Ferret version 3.1)
/FORMAT=stream is used to indicate that a file contains either unstructured binary output (typical of C program output) or fixed-length records suitable for direct access (all records of equal length, no record length information embedded in the file). With caution it is also possible to read FORTRAN variable-length record output. This sort of file is typically created by "quick and dirty" FORTRAN code which uses the simplest FORTRAN OPEN statement and outputs entire variables with a single WRITE statement.

This format specifier allows you to access any contiguous stretch of 4-byte values from the file. The /SKIP=n qualifier specifies how many values should be skipped at the file start. The /GRID=name qualifier specifies the grid onto which the data should be read and therefore the number of values to be read from the file (the number of points in the grid). Note that an attempt to read more data than the file contains, or to read record length information, will result in a fatal FORTRAN error on UNIX systems and will crash the Ferret program.

For multiple variables, use the /COLUMNS=n specifier to specify how many 4-byte values separate each variable in the file. Each variable is assumed to represent a contiguous stream of values within the file and all variables are assumed to possess the same number of points. (A "poor man's" method is to create multiple Unix soft links pointing to the same file and multiple SET DATA/EZ commands to specify one variable from each link name.)

See the chapter "Data Set Basics", section "Binary data" (p. 33) for further discussion and examples of binary types.

SET DATA/FORMAT=cdf/REGULART

Speeds initialization of large netCDF data sets with known regular time axes. When Ferret initializes a netCDF file (the SET DATA/FORMAT=cdf command or the USE command alias), it checks for the attribute point_spacing = "even" on the time axis. If found, Ferret knows that the coordinates are evenly spaced and reads only the first and last coordinate on the axis to obtain a complete description. If not found, Ferret must read the full list of coordinates -- a time-consuming procedure for very large files. After reading the coordinates, Ferret determines if they are regular. The /REGULART qualifier instructs Ferret to treat the time axis as regular regardless of the presence or absence of a point_spacing attribute in the file, speeding up the initialization time on files lacking point_spacing, but known to be regular.

Note that when writing netCDF files Ferret, by default, does NOT include the point_spacing attribute. This is because Ferret's default file characteristic is to be append-able, with no guarantees that the appended time steps will be regularly spaced. For output files of fixed

length with regular time steps it is advisable to use the SAVE/RIGID qualifier. This allows Ferret to include the point_spacing="even" attribute. If the files will be very large (too large for the full time range to be in memory), then use the /RIGID/TLIMITS= qualifiers to specify the full, ultimate fixed size and use SAVE/APPEND to insert data into the file piecemeal.

SET DATA/**RESTORE**
Restores the current default data set number that was saved with SET DATA/SAVE.

This is useful in creating GO files that perform their function and then restore Ferret to its previous state.

SET DATA/**SAVE**
Saves the current default data set number so it can be restored with SET DATA/RESTORE.

This is useful in creating GO files that perform their function and then restore Ferret to its previous state.

SET DATA/**EZ**
Accesses data from an ASCII or unformatted file that is not in a standardized format (TMAP or NetCDF). The command FILE is an alias for SET DATA/EZ.

```
yes? SET DATA/EZ[/qualifiers]   ASCII_or_binary_file
```

or, equivalently,

```
yes? FILE[/qualifiers]  ASCII_or_binary_file
```

**Example:**

```
yes? FILE/VARIABLE=my_var my_data.dat
```

See the chapter "Data Set Basics", section “ASCII data” (p. 37) for more information and examples.

**Command qualifiers for SET  DATA_SET/EZ:**

SET DATA/EZ/**COLUMNS=n**
Specifies the number of columns in the EZ data file.

By default the number of columns is assumed to be equal to the number of variables (including “-”’s) specified by the /VARIABLES qualifier.

SET DATA/EZ/**GRID=**
Specifies the defining grid for the data in the EZ data set. The argument can be the name of a grid or the name of a variable that is already defined on the desired grid.

**Example:**

```
yes? SET DATA/EZ/GRID=sst[D=coads] snoopy
```

This is the mechanism by which the shape of the data (1D along T axis, 2D in the XY plane, etc.) is specified. By default Ferret uses grid EZ, a line of up to 20480 points oriented along the X axis.

SET DATA/EZ/**ORDER=** (Ferret version 3.11)
Specifies the order (ORDER=permutation) in which axes are to be read.

**Examples:**

```
yes? FILE/ORDER=XY sst          !X varies fastest
yes? LIST/ORDER=YX sst          !Y varies fastest
```

The "permutation" string may be any permutation of the letters X, Y, Z, and T. If the /format=stream qualifier is used, the string may also contain V (for variable). This allows variables to be "interleaved."

SET DATA/EZ/**SKIP=n**
Specifies the number of records to skip at the start of an EZ data set before beginning to read the data. By default, no records are skipped.

For ASCII files a "record" refers to a single line in the file (i.e., a newline character). If the FORMAT statement contains slash characters the "data record" may be multiple lines; the /SKIP qualifier is independent of this fact.

For FORTRAN-structured binary files the /SKIP argument refers to the number of binary records to be skipped.

For unstructured (stream) binary files (e.g., output of a C program) the /SKIP argument refers to the number of words (4-byte quantities) to skip before reading begins.

SET DATA/EZ/**SWAP**
Stream files only. Change the byte ordering of numbers read from the file; big-endian numbers are converted to little-endian numbers and vice versa.

SET DATA/EZ/**TITLE=**
Associates a title with the data set.

```
yes? SET DATA/EZ/TITLE="title string"  file_name
```

This title appears on plotted outputs at the top of the plot.

SET DATA/EZ/**TYPE=**

Stream files only. Specify the data type of a set of variables in a stream file. Available values and their corresponding types are:

| Value | FORTRAN | C | size in bytes |
|---|---|---|---|
| i1 | INTEGER*1 | char | 1 |
| i2 | INTEGER*2 | short | 2 |
| i4 | INTEGER*4 | int | 4 |
| r4 | REAL*4 | float | 4 |
| r8 | REAL*8 | double | 8 |

```
yes? SET DATA/EZ/FORMAT=STREAM/TYPE=14,R4/VAR=V1,V2 foobar.dat
```

will read a file containing INTEGER*4 and REAL*4 numbers into the variables v1 and v2.

SET DATA/EZ/**VARIABLES=**
Names the variables of interest in the file. Default is v1.

```
yes? FILE/VARIABLES="var1,var2,..."  file_name
```

Except in the case of /FORMAT=stream, Ferret assumes that successive values in the data file represent successive variables. For example, if there are three variables in a file, the first value represents the first variable, the second represents the second variable, the third the third variable, and the fourth returns to representing the first variable. The maximum number of variables allowed in a single data set is 20.

Variable names may be 1 to 24 characters (letters, digits, $, and _) beginning with a letter. To indicate a column is not of interest use "-" for its name.

**Example: (the third column of data will be ignored)**

```
yes? SET DATA/EZ/VARIABLES="temp,salt,-,u,v" ocean_file.dat
```

## Ref Sec29.3. SET EXPRESSION

Specifies the default context expression. When Ferret's "action" commands (PLOT, CONTOUR, SHADE, VECTOR, WIRE, etc.) are issued with no argument, the default context expression is used. This is the expression last used as argument to an action command, or it may be set explicitly with SET EXPRESSION. See the chapter "Variables and Expressions", section "Expressions" (p. 53) for a full list of action commands.

```
yes? SET EXPRESSION expr1 , expr2 , ...
```

**Examples:**

1) `yes? SET EXPRESSION temp`
   Sets the current expression to “temp”.

2) `yes? SET EXPRESSION u , v , u^2 + v^2`
   Set the current expressions to “u , v , u^2 + v^2”

## Ref Sec29.4. SET GRID

/RESTORE /SAVE

Specifies the default grid for abstract expressions. Type `GO wire_frame_demo` at the Ferret prompt for an example of usage.

```
yes? SET GRID[/qualifier] [grid_or_variable_name]
```

**Examples:**

```
yes? SET GRID sst[D=coads]

yes? SET GRID           ! use grid from last data accessed
```

See the chapter “Grids and Regions” (p. 101).

**Command qualifiers for SET GRID:**

SET GRID/**RESTORE**
Restores the current default grid last saved by SET GRID/SAVE. Useful together with SET GRID/SAVE to create GO files that restore the state of Ferret when they conclude.

SET GRID/**SAVE**
Saves the current default grid to be restored later. Useful together with SET GRID/RESTORE to create GO files that restore the state of Ferret when they conclude.

## Ref Sec29.5. SET LIST

/APPEND /FILE /FORMAT /HEADING /PRECISION

Uses SET LIST to specify the default characteristics of listed output.

```
yes? SET LIST/qualifiers
```

The state of the list command may be examined with SHOW LIST. See command CANCEL LIST (p. 243) and LIST (p. 270).

**Command qualifiers for SET LIST:**

SET LIST/**APPEND**
Specifies that by default the listed output is to be appended to a pre-existing file. Cancel this state with CANCEL LIST/APPEND.

SET LIST/**FILE=**
Specifies a default file for the output of the LIST command.

```
yes? SET LIST/FILE=filename
```

The filename specified in this way is a default only. It will be used by the command

```
yes? LIST/FILE variable
```

but will be ignored in

```
yes? LIST/FILE=snoopy.dat variable
```

Ferret generates a filename based on the data set, variable, and region if the filename specified is "AUTO". The resulting name is often quite long but may be shortened by following "AUTO" with a minus sign and the name(s) of the axes to exclude from the filename.

**Note:** the region information is not used in automatic NetCDF output filenames.

**Examples:**

```
yes? SET LIST/FILE=AUTO
yes? LIST/L=500/X=140W:110W/Y=2S:2N/FILE sst[D=coads]
```

Sends data to file WcoadsSST.X140W110WY2S2NL500.

```
yes? SET LIST/FILE=AUTO-XY
yes? LIST/L=500/X=140W:110W/Y=2S:2N/FILE sst[D=coads]
```

Sends data to file WcoadsSST.L500.

SET LIST/**FORMAT=**
Specifies an output format for the LIST command. (When a FORTRAN format is specified the row and column headings are omitted from the output.)

```
yes? SET LIST/FORMAT=option
yes? SET LIST/FORMAT        !reactivate previous format
```

**Options**

FORTRAN format  produces ASCII output

| | |
|---|---|
| "UNFORMATTED" | produces unformatted (binary) output |
| "CDF" | produces NetCDF output |
| "GT" | produces TMAP GT format |

**Examples:**

1) `yes? SET LIST/FORMAT=(1X,12F6.1)`
Specifies a FORTRAN format (without row or column headings).

2) `yes? SET LIST/FORMAT=UNFORMATTED`
Specifies binary output. (FORTRAN variable record length record structure.)

**Notes:**

- When using GT format all variables named in a single LIST command will be put into a single GT-formatted timestep.
- Very limited error checking will be done on FORTRAN formats.
- FORTRAN formats are reused as necessary to output full record.
- Latitude axes are listed south to north when /FORMAT is specified.

SET LIST/**HEAD**
Specifies that ASCII output is to be preceded by a heading that documents data set, variable, and region. Cancel the heading with CANCEL LIST/HEAD.

SET LIST/**PRECISION**
Specifies the data precision (number of significant digits) of the output listings. This qualifier has no effect when /FORMAT= is specified.

```
yes? SET LIST/PRECISION=#_of_digits
```

## Ref Sec29.6. SET MEMORY

/SIZE

```
yes? SET MEMORY/SIZE=megawords
```

The command SET MEMORY provides control over how much "physical" memory Ferret can use. (In reality the distinction between physical and virtual memory is invisible to Ferret. The SET MEMORY command merely dictates how much memory Ferret can attempt to allocate from the operating system.)

SET MEMORY controls only the size of Ferret's cache memory—memory used to hold intermediate results from computations that are in progress and used to hold the results of past file IO and computations for re-use. The default size of the memory cache is 3.2 megawords

(equivalently, 3.2×4=12.8 megabytes). Cache memory size can be set larger or smaller than this figure.

**Example:**

```
yes? SET MEMORY/SIZE=4.2
```

Sets the size of Ferret's memory cache to 4.2 million (4-byte) words.

**Notes:**

- As a practical matter memory size should not normally be set larger than the physical memory available on the system.
- The effect of SET MEMORY/SIZE= is identical to the "-memsize" qualifier on the Ferret command line.
- See SET MODE DESPERATE (p. 302) and MEMORY USAGE (p. 187) in this users guide for further instructions on setting the memory cache size appropriately.
- Using the SET MEMORY command automatically resets the value of SET MODE DESPERATE to a default that is consistent with the memory size.
- The effects of SET MEMORY/SIZE last only for the current Ferret session. Exiting Ferret and restarting will reset the memory cache to its default size.
- If memory is severely limited on a system Ferret's default memory cache size may be too large to permit execution. In this case use the "-memsize" qualifier on the command line to specify a smaller cache.

## Ref Sec29.7. SET MODE

/LAST

Specifies special operating modes or states for program Ferret.

```
yes? SET MODE[/LAST] mode_name[:argument]
```

| Mode | Description | Default State |
|---|---|---|
| ASCII_FONT | imposes PPLUS ASCII font types on plot labels | set |
| CALENDAR | uses date strings for T axis (vs. time step values) | set |
| DEPTH_LABEL | uses "DEPTH" as Z axis label | set |
| DESPERATE | attempts calculations too large for memory | canceled |
| DIAGNOSTIC | turns on internal program diagnostic output | canceled |
| GUI | unsupported; used in GUI development | |
| IGNORE_ERROR | continues command file after errors | canceled |
| INTERPOLATE | automatically interpolates data between planes | canceled |

| Mode | Description | Default State |
|---|---|---|
| JOURNAL | records keyboard commands in a journal file | set |
| LATIT_LABEL | uses “N” “S” notation for labeling latitudes | set |
| LONG_LABEL | uses “E” “W” notation for labeling longitudes | set |
| METAFILE | captures graphics in GKS metafiles | canceled |
| POLISH | interprets expressions in Reverse Polish order | canceled |
| PPLLIST | listed output from PPLUS is directed to the named file | canceled |
| REFRESH | refreshes graphics on systems lacking “backing store” | canceled |
| SEGMENT | utilizes GKS segment storage | set |
| STUPID | controls cache hits in memory (diagnostic) | canceled |
| VERIFY | displays each command file line as it is executed | set |
| WAIT | waits for carriage return after each plot | canceled |

**Command qualifiers for SET MODE:**

SET MODE/**LAST**
Resets mode to its last state.

```
yes? SET MODE/LAST mode_name
```

**Example: (a command file that will not alter Ferret modes)**

```
yes? SET MODE IGNORE_ERRORS     ! 1st line of command file
       .
       . ... code which may encounter errors
       .
yes? SET MODE/LAST IGNORE_ERRORS ! last line of command file
```

## Ref Sec29.7.1. SET MODE ASCII_FONT

The SET MODE ASCII_FONT command causes program Ferret to precede plot labels with the PPLUS font descriptor “@AS” (ASCII SIMPLEX font). This assures that special characters (e.g., underscores) are faithfully reproduced. For special plots it may be desirable to use other fonts (e.g., to obtain subscripts). CANCEL MODE ASCII_FONT is for these cases.

default state: set

## Ref Sec29.7.2. SET MODE CALENDAR

SET MODE CALENDAR causes program Ferret to output times in date/time format (instead of time axis time step values). This affects both plotted and listed output.

This mode accepts an optional argument specifying the degree of precision for the output date. If the argument is omitted the precision is unchanged from its last value.

default state: set (argument: minutes)

**Arguments**

SET MODE CALENDAR accepts the following arguments:

| Argument | Equivalent precision |
|---|---|
| SECONDS | -6 |
| MINUTES | -5 (default) |
| HOURS | -4 |
| DAYS | -3 |
| MONTHS | -2 |
| YEARS | -1 |

The argument is uniquely identified by the first two characters.

**Example:**

```
yes? SET MODE CALENDAR:DAYS
```

Causes times to be displayed in the format dd-mmm-yyyy.

When CALENDAR mode is canceled the "equivalent" in the table above determines the precision of the time steps displayed exactly as in SET MODE LONGITUDE.

## Ref Sec29.7.3. SET MODE DEPTH_LABEL

SET MODE DEPTH_LABEL causes Ferret to label Z coordinate information in the units of the Z axis. This affects both plotted and listed output. This mode accepts an optional argument specifying the degree of precision for the output. If the argument is omitted the precision is unchanged from its last value.

```
yes? SET MODE DEPTH:argument
```

default state: set (argument: -4)

**Arguments**

See SET MODE LONG (p. 304) for a detailed description of precision control.

## Ref Sec29.7.4. SET MODE DESPERATE

Ferret checks the size of the component data required for a calculation in advance of performing the calculation. If the size of the component data exceeds the value of the MODE DESPERATE argument Ferret attempts to perform the calculation in pieces.

For example, the calculation "LIST/I=1/J=1 U[K=1:100,L=1:1000@AVE]" requires 100*1000=100,000 points of component data although the result is only a line of 100 points on the K axis. If 100,000 exceeds the current value of the MODE DESPERATE argument Ferret splits this calculation into smaller sized chunks along the K axis, say, K=1:50 in the first chunk and K=51:100 in the second.

Ferret is also sensitive to the performance penalties associated with reading data from the disk. Splitting the calculation along axis of the stored data records can require the data to be read many times in order to complete the calculation. Ferret attempts to split calculations along efficient axes, and will split along the axis of stored data only in desperation, if MODE DESPERATE is SET.

**Example:**

```
yes? SET MODE DESPERATE:5000
```

default state: canceled (default argument: 80000)

**Note:** Use MODE DIAGNOSTIC to see when splitting is occurring.

**Arguments**

Use SHOW MEMORY/FREE to see the total memory available (as set with SET MEMORY/SIZE).

Whenever the size of memory is set using SET MEMORY the MODE DESPERATE argument is reset at one tenth of memory size. For most purposes this will be an appropriate value. The user may at his discretion raise or lower the MODE DESPERATE value based on the nature of a calculation. A complex calculation, with many intermediate variables, may require a smaller value of MODE DESPERATE to avoid an "insufficient memory" error. A simple calculation, such as the averaging operation described above, will typically run faster with a larger MODE DESPERATE value. The upper bound for the argument is the size of memory. The lower bound is "memory block size."

## Ref Sec29.7.5. SET MODE DIAGNOSTIC

SET MODE DIAGNOSTIC causes Ferret to display diagnostic information in real time about its internal functioning. It is intended to help Ferret developers diagnose performance problems by displaying what the Ferret memory management subsystem is doing. The message

"strip gathering on xxx axis" indicates that Ferret has broken up a calculation into smaller pieces. Subsequent "strip" and "gathering" messages indicate that sub-regions of the calculations are being processed and brought together.

default state: canceled

## Ref Sec29.7.6. SET MODE IGNORE_ERROR

SET MODE IGNORE_ERROR causes Ferret to continue execution of a command file despite errors encountered. (See command GO, p. 266.)

default state: canceled

## Ref Sec29.7.7. SET MODE INTERPOLATE

**Note:** The transformation @ITP provides the same functionality as MODE INTERPOLATE with a greater level of control.

SET MODE INTERPOLATE affects the interpretation of world coordinate specifiers (/X, /Y, /Z, and /T) in cases where the position is normal to the plane in which the data is being examined. When this mode is SET and a world coordinate is specified which does not lie exactly on a grid point, Ferret automatically interpolates from the surrounding grid point values. When this mode is canceled, the same world coordinate specification is shifted to the grid point of the grid box that contained it before computations were made (see examples).

default state: canceled

**Example:**

If the grid underlying the variable temp has points defined at Z=5 and at Z=15 (with the grid box boundary at Z=10) and data is requested at Z=12 then

```
yes? SET MODE INTERPOLATE
yes? LIST/T=18249/X=130W:125W/Y=0:3N/Z=12 temp
```

lists temperature data in the X-Y plane obtained by interpolating between the Z=5 and Z=15 planes. Whereas,

```
yes? CANCEL MODE INTERPOLATE
yes? LIST/T=18249/X=130W:125W/Y=0:3N/Z=12 temp
```

lists the data at Z=15. The output documentation always reflects the true location used.

## Ref Sec29.7.8. SET MODE JOURNAL

SET MODE JOURNAL causes Ferret to record all commands issued in a journal file. Output echoed to this file may be turned on and off via mode JOURNAL at any time.

default state: set

**Example:**

```
yes? SET MODE JOURNAL:my_journal_file.jnl
```

The optional argument to MODE JOURNAL specifies the name of the output journal file—with no argument, the default name "ferret.jnl" is used. Journal files for successive Ferret sessions are handled by version number. See the chapter "Computing Environment", section "Output file naming" (p. 191).

## Ref Sec29.7.9. SET MODE LATIT_LABEL

SET MODE LATIT_LABEL causes Ferret to output latitude coordinate information in degrees N/S format (instead of the internal latitude coordinate). This affects both plotted and listed output.

This mode accepts an optional argument specifying the degree of precision for the output. If the argument is omitted the precision is unchanged from its last value.

**Example:**

```
yes? SET MODE LAT:2
```

default state: set (argument: 1)

**Arguments**

See command SET MODE LONG (p. 304) for a detailed description of precision control.

## Ref Sec29.7.10. SET MODE LONG_LABEL

SET MODE LONG_LABEL causes Ferret to output longitude coordinate information in degrees E/W format (instead of the internal longitude coordinate). This will affect both plotted and listed output.

This mode accepts an optional argument specifying the degree of precision for the output. If the argument is omitted the precision will be unchanged from its last value.

**Example:**

```
yes? SET MODE LONG:2
```

default state: set (argument: 1)

**Arguments**

The argument of SET MODE LONG is an integer specifying the precision. If the argument is positive or zero it specifies the maximum number of decimal places to display. If the argument is negative it specifies the maximum number of significant digits to display.

**Examples:**

Suppose the longitude to be displayed is 165.23W. Then

```
yes? SET MODE LONG:1    will produce 165.2W
yes? SET MODE LONG:-3   will produce 165W
```

When LONG mode is canceled the argument still determines the output precision.

## Ref Sec29.7.11. SET MODE METAFILE

SET MODE METAFILE causes Ferret to capture all graphics in metafiles. These metafiles can later be routed to various devices to obtain hard copy output.

The optional argument to MODE METAFILE specifies the name of the output metafile—with no argument, the default name “metafile.plt” is used. Multiple output files (i.e., successive plots) are handled by version number. See the chapter "Computing Environment", section “Output file naming” (p. 191).

See the chapter "Computing Environment", section “Hard copy” (p. 188) for details on generating hard copy.

**Example:**

```
yes? SET MODE METAFILE:june_sst.plt
```

default state: canceled (default argument when set: “metafile.plt”)

## Ref Sec29.7.12. SET MODE POLISH

The SET MODE POLISH command causes program Ferret to expect algebraic expressions to be entered in Reverse Polish order.

This mode exists only to assist with compatibility with earlier versions of Ferret. It has no efficiency advantages.

default state: canceled

## Ref Sec29.7.13. SET MODE PPLLIST

Directs listed output from PPLUS commands (e.g., PPL LIST LABS) to the specified file. This mode is useful for creating scripts that customize plots. The user can specify the name of the output file by giving it as an argument, otherwise file name "ppllist.out" is assigned.

**Example:**

```
yes? SET MODE PPLLIST:plot_symbols.txt
yes? PPL LISTSYM
yes? SPAWN grep "WIDTH" plot_symbols.txt
```

default state: canceled

## Ref Sec29.7.14. SET MODE REFRESH

The SET MODE REFRESH command causes Ferret to update windows following "occlusion" events on X-servers that lack a backing store (SGI workstations have been a case in point).

default state: canceled (except on SGI systems)

## Ref Sec29.7.15. SET MODE SEGMENTS

SET MODE SEGMENTS causes Ferret to utilize GKS segments ("GKS" is the Graphical Kernel System—an international graphics standard). On some systems MODE SEGMENTS may be necessary to update windows following "occlusion" events or to resize window with the mouse.

Segments, however, make heavy demands on the system's virtual memory. If Ferret crashes during graphics output due to insufficient virtual memory try CANCEL MODE SEGMENTS.

default state: set

## Ref Sec29.7.16. SET MODE STUPID

**Note:** MODE STUPID is included for diagnostic purposes only.

SET MODE STUPID controls the ability of Ferret to reuse results left in memory from previous commands. It also effects its ability to reuse intermediate variables that are referenced multiple times during complex calculations. Given with no argument

```
yes? SET MODE STUPID
```

causes Ferret to forget data cached in memory. The result is that all requests for variables are read from disk and no intermediate calculations are reused. The program will be significantly slower as a result.

A lesser degree of cache limitation occurs with the command

```
yes? SET MODE STUPID: weak_cache
```

which causes Ferret to revert to the cache access strategy that it used previous to Ferret version 5.0. In this mode cache hits are unreliable unless the region of interest is fully specified. (Unspecified limits will typically default to the full range of the relevant axis.)

default state: canceled

## Ref Sec29.7.17. SET MODE VERIFY

SET MODE VERIFY causes commands from a command file ("GO file") to be displayed on the screen as they are executed. Note that if MODE VERIFY is canceled, loop counting in the REPEAT command is turned off.

default state: SET, argument "default"

**Note:** Many GO files begin with CANCEL MODE VERIFY to inhibit output and end with SET MODE/LAST VERIFY to restore the previous state. Only if an error or interrupt occurs during the execution of such a command file will the state of MODE VERIFY be affected.

SET MODE VERIFY can accept arguments to further refine control over command echoing.

```
yes? SET MODE VERIFY: DEFAULT
```

- This will be the default state if no argument is given
- Ferret echos commands taken from GO scripts
- Ferret echos commands in which symbol substitutions occur or in which embedded expressions are evaluated

```
yes? SET MODE VERIFY: ALL
```

- In addition to the cases above Ferret also displays the individual commands that are generated by repeat loops and semicolon-separated command groups
- Ferret displays a REPEAT loop counter ("!-> REPEAT: ...")

```
yes? SET MODE VERIFY: ALWAYS
```

- Echoing behavior is the same as argument ALL but ALWAYS, in addition, causes CANCEL MODE VERIFY to be ignored when it is encountered in a GO file. This functionality is useful when debugging GO scripts. Entering CANCEL MODE VERIFY or SET MODE VERIFY:DEFAULT from the command line will cancel this state.

## Ref Sec29.7.18. SET MODE WAIT

SET MODE WAIT causes Ferret to wait for a keyboard keystroke from the user after each plotted output is completed. This is useful on graphics terminals that do not have a separate graphics plane; on these terminals SET MODE WAIT prevents the graphical output from being wiped off the screen until the user is ready to proceed.

default state: canceled

## Ref Sec29.8. SET MOVIE

/COMPRESS /FILE /LASER /START

Designates a file (specified or default) for storing graphical images as movie frames (in HDF Raster-8 format). Note that the FRAME/FILE=filename qualifier is generally preferable to the SET MOVIE command, as it is simpler and more flexible. See the chapter "Animations and GIF Images (p. 123) for further explanation.

```
yes? SET MOVIE[/qualifiers]
```

**Command qualifiers for SET MOVIE:**

SET MOVIE/**COMPRESS=**
Turns on or off compression of HDF frames using run length compression.

```
yes? SET MOVIE/COMPRESS=OFF
```

The allowed arguments are "on" and "off" —CANCEL MOVIE does not affect this qualifier.

default state: on

SET MOVIE/**FILE**
Specify an output file to receive movie frames.

```
!specify a new filename
yes?  SET MOVIE/FILE=filename
```

or

```
!reactivate a previously specified filename after CANCEL MOVIE
yes?  SET MOVIE/FILE
```

The default movie filename extension is ".mgm"

The default movie filename is "ferret.mgm"

SET MOVIE/**LASER**
Output to Panasonic OMDR. Valid only on older VAX/VMS systems.

SET MOVIE/**START**
Only valid for use on older VAX/VMS systems with the Panasonic Optical Memory Disk Recorder (OMDR). Only valid with /LASER qualifier.

## Ref Sec29.9. SET REGION

/I/J/K/L /X/Y/Z/T /DI/DJ/DK/DL /DX/DY/DZ/DT

Specifies the default space-time region for the evaluation of expressions.

```
yes? SET REGION[/qualifiers] [ reg_name]
```

See the chapter "Grids and Regions", section "Regions" (p. 116) for further information.

**Examples:**

1) `yes? SET REGION/X=140E`
   Sets X axis position in the default context.

2) `yes? SET REGION/@N     !N specifies X and Y but not Z or T`
   Sets only X and Y in the default context (since X and Y are defined in region N but Z and T are not).

3) `yes? SET REGION N`
   Sets ALL AXES in the default region to be exactly the same as region N. Since Z and T are undefined in region N they will be set undefined in the default context.

4) `yes? SET REGION/@N/Z=50:250`
   Sets X and Y in the default region to be exactly the same as region N and then sets Z to the range 50 to 250.

5) `yes? SET REGION/DZ=-5`
   Set the region along the Z axis to be 5 units less than its current value.

6) `yes? SET REGION/DJ=-10:10`
Increases the current vertical axis range by 10 units on either end of the axis.

**Command qualifiers for SET REGION:**

SET REGION/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Sets region bounds for specified axis subscript (I, J, K, or L) or axis coordinates (X, Y, Z, or T). See examples above.

SET REGION/**DI=/DJ=/DK=/DL=/DX=/DY=/DZ=/DT=**
Modifies current region information by the specified increment of an axis subscript (I, J, K, or L) or axis coordinate (X, Y, Z, or T). See examples above. Syntax: /D*=val, or /D*=lo:hi.

## Ref Sec29.10. SET VARIABLE

/BAD /GRID /TITLE /UNIT /DATASET

Modifies attributes of a variable defined by DEFINE VARIABLE or SET DATA/EZ. This command permits variables within a single EZ data set to be defined on different grids and it allows the titles and units to be superseded for the duration of a session, only, on NetCDF and GT data sets.

```
yes? SET VARIABLE/qualifiers variable_name
```

**Parameters**

The variable name can be a simple name or a name qualified by a data set.

**Example:**

```
yes? SET VAR/UNITS="CM" WIDTH[D=snoopy]
```

**Command qualifiers for SET VARIABLE:**

SET VARIABLE/**BAD=**
Designates a value to be used as the missing data flag. The qualifier is applicable to EZ data set variables and to NetCDF data sets. The bad value which is specified will be used in subsequent outputs and calculations involving this variable. It applies only for the duration of the current Ferret session. It does not alter the data files. It is not applicable to variables defined with DEFINE VARIABLE.

When the command SET VARIABLE/BAD= is be used to set one of the two missing value flags of a file variable, the bad value which is specified will be used in subsequent outputs and calculations involving this variable

Ferret is aware of up to two missing value flags for each variable in a netCDF file. Under most circumstances, netCDF file variables use only a single flag. With a command like

```
SET VARIABLE/BAD=-999 my_file_var
```

you can specify -999 as an additional missing value flag. It is this value which will be present in all subsequent SAVEs to files and calculations.

Note that if the netCDF file contains two distinct flag values specified by the netCDF attributes "missing_value" and "_FillValue", then this command will migrate the value specified by missing_value to the position previously occupied by _FillValue and replace the one specified by missing_value. Thus a double usage of this command allows you to control both flags. You can use the command "SHOW DATA/VARIABLES" to see both bad value flags.

SET VARIABLE/**GRID=**
Sets the defining grid for a variable in an EZ data set.

**Example:**

```
yes? SET VARIABLE/GRID=my_grid width[D=snoopy]
```

This is the mechanism by which the shape of the data (1D along T axis, 2D in the XY plane, etc.) is specified. By default Ferret will use grid EZ, a line of up to 20480 points oriented along the X axis. The qualifier is not applicable to variables defined with DEFINE VARIABLE.

SET VARIABLE/**TITLE=**
Associates a title with the variable. This title appears on plotted outputs and listings. The qualifier is applicable to all variables.

```
yes? SET VARIABLE/TITLE="title string" var_name
```

SET VARIABLE/**UNITS=**
Associates units with the variable. The units appear on plotted outputs and listings. The qualifier is applicable to all variables.

```
yes? SET VARIABLE/UNITS="units string" var_name
```

## Ref Sec29.11. SET VIEWPORT

Sets the rectangular region within the output window where output will be drawn.

```
yes? SET VIEWPORT view_name
```

Issuing the command SET VIEWPORT is best thought of as entering “viewport mode.” While in viewport mode all previously drawn viewports remain on the screen until explicitly cleared with either SET WINDOW/CLEAR or CANCEL VIEWPORT. If multiple plots are drawn in a

single viewport without the use of /OVERLAY the current plot will erase and replace the previous one; the graphics in other viewports will be affected only if the viewports overlap. If viewports overlap the most recently drawn graphics will always lie on top, possibly obscuring what is underneath. By default, the state of "viewport mode" is canceled.

Pre-defined viewports exist for dividing the window into four quadrants and for dividing the window in half horizontally and vertically. See the chapter "Customizing Plots", section "Pre-defined viewports" (p. 151) for a list.

## Ref Sec29.12. SET WINDOW

/ASPECT /CLEAR /LOCATION /NEW /SIZE

Creates, resizes, reshapes or moves graphics output windows.

```
yes? SET WINDOW[/qualifiers]  [window_number]
```

**Note:** Multiple windows may be simultaneously viewable but only a single window receives output at any time.

See commands SHOW WINDOWS (p. 326) and CANCEL WINDOW (p. 247) for additional information.

**Examples:**

1) `yes? SET WINDOW/NEW`
   Creates a new output window and sends subsequent graphics to it.

2) `yes? SET WINDOW 3`
   Sends subsequent graphics to window 3.

3) `yes? SET WINDOW/SIZE=.5`
   Resizes current window to ½ of full.

4) `yes? SET WINDOW/ASPECT=.5`
   Reshapes current window with Y/X equal to 1:2.

5) `yes? SET WINDOW/LOCATION=0,.5`
   Puts the lower left corner of the current window at the left border of the display and half way up it.

**Command qualifiers for SET WINDOW:**

SET WINDOW/**ASPECT**
Sets the aspect ratio of the output window and hard copy.

**Examples:**

1) `yes? SET WINDOW/ASPECT=y_over_x  n`
   Sets the overall aspect ratio of window n.

2) `yes? SET WINDOW/ASPECT=y_over_x`
   Sets the overall aspect ratio of the current window.

3) `yes? SET WINDOW/ASPECT=y_over_x:AXIS`
   Sets the axis length aspect ratio of the current window.

The total size (area) of the output window is not changed.
The default value for the overall window ratio is y/x = 8.8/10.2 ~ 0.86.
The default value for the axis length ratio is y/x = 6/8 = 0.75.
Use PPLUS/RESET or SET WINDOW/ASPECT=.75:AXIS to restore defaults.
The aspect ratio specified is a default for future SET WINDOW commands
The origin (lower left) is restored to its default values:  1.2, 1.4

When using "SET WINDOW n" to return to a previous window that differs from the current window in aspect ratio, it is necessary to re-specify its aspect ratio with /ASPECT, otherwise PPLUS will not be properly reset.

SET WINDOW/**CLEAR**
Clears the image(s) in the current or specified window. Useful with viewports.

SET WINDOW/**LOCATION**
Sets the location for the lower left corner of named (or current) window. The coordinates x and y must be values between 0 and 1 and refer to distances from the lower left corner of the display screen (total length and width of which are each 1).

```
yes? SET WINDOW/LOCATION=x,y [window_number]
```

SET WINDOW/**NEW**
Causes future graphical output to be directed to a new window. The window will be created at the next graphics output.

```
yes? SET WINDOW/NEW
```

SET WINDOW/**SIZE**
Resizes a window to r times the size of the standard window. If the window number is omitted the command will resize the currently active window. (The default window size is 0.7.)

```
yes? SET WINDOW/SIZE=r [window_number]
```

## Ref Sec30. SHADE

/I/J/K/L /X/Y/Z/T /D /FRAME /KEY /LEVELS /LINE /NOAXIS /NOKEY /NOLABELS /OVERLAY /PALETTE /PATTERN /SET_UP /TITLE /TRANSPOSE /HLIMITS/ /VLIMITS /XLIMITS /YLIMITS

Produces a shaded (rectangular raster) plot of a 2-D field. By default a color key is drawn and contour lines are not drawn.

```
SHADE[/qualifiers]  expression
```

**Parameters**

The expression may be any valid expression. See the chapter "Variables and Expressions", section “Expressions” (p. 53) for a definition of valid expressions. The expression will be inferred from the current context if omitted from the command line. Multiple expressions are not permitted in a single SHADE command.

**Command qualifiers for SHADE:**

SHADE/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression being plotted.

SHADE/**D=**
Specifies the default data set to be used when evaluating the expression being plotted.

SHADE/**FRAME**
Causes the graphic image produced by the command to be captured as an animation frame in the file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

SHADE/**KEY**
Displays a color key for the palette used in the shaded plot. By default a key is drawn unless the /LINE or /NOKEY qualifier is specified.

SHADE/**LEVELS**
Specifies the SHADE levels or how the levels will be determined. If the /LEVELS qualifier is omitted Ferret automatically selects reasonable SHADE levels.

See the chapter "Customizing Plots", section “Contouring” (p. 154) for examples and more documentation on /LEVELS and color_thickness indices, and also the demonstration “custom_contour_demo.jnl”.

SHADE/**LINE**
Overlays contour lines on a shaded plot. When /LINE is specified the color key is omitted unless specifically requested via /KEY.

SHADE/**NOKEY**
Suppresses the drawing of a color key for the palette used in the plot.

SHADE/**NOAXIS**
Suppresses all axis lines, tics and labeling so that no box appears surrounding the contour plot. This is especially useful for map projection plots.

SHADE/**NOLABELS**
Suppresses all plot labels except axis labels.

SHADE/**OVERLAY**
Causes the indicated shaded plot to be overlaid on the existing plot.

Note (SHADE/OVERLAY with time axes):
A restriction in PPLUS requires that if time is an axis of the shaded plot, the overlaid variable must share the same time axis encoding as the base plot variable. If this condition is not met, you may find that the overlaid shaded plot fails to be drawn. The solution is to use the Ferret regridding capability to regrid the base plot variable and the overlaid plot variable onto the same time axis.

SHADE/**PALLETTE=**
Specifies a color palette (otherwise, a default rainbow palette is used). Try the Unix command `% Fpalette '*'` to see available palettes. The file suffix *.spk is not necessary when specifying a palette. See command PALETTE (p. 276) for more information.

```
Yes? SHADE/PALETTE=land_sea   rose
```

The /PALETTE qualifier changes the current palette for the duration of the plotting command and then restores the previous palette. This behavior is not immediately compatible with the /SET_UP qualifier. See the PALETTE (p. 276) command for further discussion.

SHADE/**PATTERN=**
Specifies a pattern file (otherwise, the current default pattern specification is used). The file suffix *.pat is not necessary when specifying a pattern. Try the Unix command `% Fpattern '*'` to see available patterns. See command PATTERN (p. 277) for more information.

SHADE/**SET_UP**
Performs all the internal preparations required by program Ferret for a shaded plot but does not actually render output. The command PPL can then be used to make changes to the plot prior to producing output with the PPL SHADE command. This permits plot customizations that are not possible with Ferret command qualifiers. See the chapter "Customizing Plots" (p. 129).

SHADE/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about the expression(s). To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character. See the chapter "Customizing Plots", section “Fonts” (p. 149).

```
yes? SHADE/TITLE="title string"  expression
```

SHADE/**TRANSPOSE**
Causes the horizontal and vertical axes to be interchanged. By default the X axis is drawn horizontally on the plot and the Y and Z axes are drawn vertically. For Y-Z plots the Z data axis is vertical.

Note that plots in the YT and ZT planes have /TRANSFORM applied by default in order to achieve a horizontal T axis. See /XLIMITS (below) for further details. Use /TRANSPOSE manually to reverse this effect.

SHADE/**HLIMITS=**
Specifies the horizontal axis range and tic interval (otherwise, Ferret selects reasonable values).

```
yes? SHADE/HLIMITS=lo:hi:increment
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

SHADE/**VLIMITS=**
Specifies the vertical axis range and tic interval. See /HLIMITS (above)

SHADE/**XLIMITS=**
Note: **XLIMITS** and **YLIMITS** have been denigrated. Please use **HLIMITS** and **VLIMITS** instead.

Specifies the X axis range and tic interval (otherwise, Ferret selects reasonable values).

```
yes? SHADE/XLIMITS=lo:hi:increment
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

Note that the “X” in /XLIMITS refers to the horizontal axis of the plot rather than to the X axis of the grid. This can lead to confusion, especially on plots in the YT or ZT plane. Plots in these planes are automatically transposed to place the Y or Z axis, respectively, on the vertical axis of the plot. Plots may also be transposed manually with the /TRANSPOSE qualifier. On transposed plots /XLIMITS will refer to the vertical axis of the plot.

SHADE/**YLIMITS=**
Specifies the Y axis range and tic interval. See /XLIMITS (above).

## Ref Sec31. SHOW

/ALL

Displays program states and stored values.

**Command qualifiers for SHOW:**

SHOW/**ALL**
Executes all SHOW options. This command gives a complete description of the current state, including information about region, grids, axes, variables, and the state of various modes (default or set with SET MODE).

```
yes? SHOW/ALL
```

**Arguments:**

The names of variables, data sets, or other definitions can be specified using wildcards. The * wildcard matches any number of characters in the name; the question wildcard matches exactly one character.

## Ref Sec31.1. SHOW ALIAS

Lists all command aliases and the full command names for which they stand, or, with an argument, shows a specified command alias.

```
yes? SHOW ALIAS  [alias_name]
```

## Ref Sec31.2. SHOW AXIS

Shows a basic description of the named axis.

```
SHOW AXIS[/qualifiers]   axis_name
```

A typical output appears below. The columns are:

| | |
|---|---|
| name | name of axis (used also in DEFINE AXIS and DEFINE GRID) |
| axis | the orientation of the axis; “(-)” on a depth axis indicates increasing downward |

| | |
|---|---|
| # pts | number of points on axis; “r” or “i” for regular or irregular spacing, “m” if the axis is “modulo” (repeating) |
| start | position of first point on the axis |
| end | position of last point on the axis |

```
yes? SHOW AXIS PSXT
name       axis              # pts   start                end
PSXT       LONGITUDE         160 r   130.5E               70.5W
```

**Command qualifiers for SHOW AXIS:**

SHOW AXIS/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Displays the coordinates and grid box sizes for the specified axis. Optionally, low and high limits and a delta value may be specified to restrict the range of values displayed.

```
yes? SHOW AXIS/X[=lo:hi:delta] axis-name
```

**Example:**

```
yes? SHOW AXIS/L=1:12:3  my_custom_time_axis
```

SHOW/**ALL**
Show a brief summary of all axes defined.

```
yes? SHOW AXIS/ALL
```

## Ref Sec31.3. SHOW COMMANDS

Displays commands, subcommands, and qualifiers recognized by program Ferret. This command does not display aliases; use SHOW ALIAS.

```
SHOW COMMAND    [command_name or partial_command]
```

**Note:** This is the most reliable way to view command qualifiers. The output of this command will be current even when this manual is out of date.

**Examples:**

```
yes? SHOW COMMAND S       ! show all commands beginning with "S"
yes? SHOW COMMAND         ! show all commands
yes? SHOW COMMAND PLOT    ! shows command PLOT and all its qualifiers
```

## Ref Sec31.4. SHOW DATA_SET

/ALL /BRIEF /FILES /FULL /VARIABLE

Shows information about the data sets which have been SET and indicates the current default data set. By default the variables and their subscript ranges are also listed.

```
yes? SHOW DATA[/qualifiers]   [set_name_or_number1,set2,...]
```

If no data set name or number is specified then all SET data sets are shown.

**Command qualifiers for SHOW DATA_SET:**

SHOW DATA/**ALL**
This qualifier has no effect on this command; it exists for compatibility reasons.

SHOW DATA/**BRIEF**
Shows only the names of the data sets; does not describe the data contained in them.

SHOW DATA/**FILES**
Displays the names of the data files for this data set and the ranges of time steps contained in each. Output is formatted as date strings or as time step values depending on the state of MODE CALENDAR.

SHOW DATA/**FULL**
Equivalent to /VARIABLES and /FILES used together.

SHOW DATA/**VARIABLES**
In addition to the information given by the SHOW DATA command with no qualifiers, this query also provides the grid name and world coordinate limits for each variable in the data set.

**Example: SHOW DATA**

SHOW DATA produces a listing similar to the one below. The output begins with the descriptor file name (for TMAP-formatted data) and data set title. The columns I, J, K, and L give the subscript limits for each variable with respect to its defining grid (use SHOW DATA/FULL and SHOW GRID variable_name for more information).

```
yes? SET DATA levitus_climatology
yes? SHOW DATA
     currently SET data sets:
    1> /home/e1/tmap/fer_dsets/descr/levitus_climatology.des  (default)
      name     title                I         J         K         L
      TEMP     TEMPERATURE        1:360     1:180     1:20      1:1
      SALT     SALINITY           1:360     1:180     1:20      1:1
```

## Ref Sec31.5. SHOW EXPRESSION

Shows the current expression(s) implied or set with SET EXPRESSION. If not explicitly set with this command, the default current context expression is the argument of the most recent "action" command (PLOT, SHADE, CONTOUR, VECTOR, WIRE, etc.) See the chapter

"Variables and Expressions", section “Expressions” (p. 53) for an explanation and list of action commands.

```
yes? SHOW EXPRESSION
```

## Ref Sec31.6. SHOW FUNCTION

/ALL /BRIEF /EXTERNAL /INTERNAL /DETAILS

Shows a complete list of the functions defined in Ferret including descriptions of the function arguments.

```
yes? SHOW FUNCTION[/qualifiers] [function_name]
```

If no qualifier or function name is given then all functions are listed. SHOW FUNCTION will accept name templates such as

```
yes? SHOW FUNCTION *day*
     DAYS1900 (day, month, year)
     days elapsed since Jan. 1, 1900
```

**Parameters**

The parameter(s) may be the name of a function, with * replacing part of the string as above.

**Command qualifiers for SHOW FUNCTION:**

SHOW FUNCTION/**ALL**
List all functions defined

SHOW FUNCTION/**BRIEF**
List the functions and their arguments in brief form; no function or argument descriptions.

SHOSHOW FUNCTION/**EXTERNAL**
List only the available Ferret external functions (p. 193).

SHOW FUNCTION/**INTERNAL**
List only the internally defined Ferret functions.

SHOW FUNCTION/**DETAILS**
Lists the axis inheritance for grid-changing functions

**Example:.SHOW FUNCTION/DETAILS**

```
yes? SHOW FUNCTION/DETAILS SAMPLEXY
SAMPLEXY(DAT_TO_SAMPLE,XPTS,YPTS)
    Returns data sampled at a set of (X,Y) points, using linear
```

```
interpolation
        Grid of result:
          X: ABSTRACT (result will occupy indices 1...N)
          Y: NORMAL (no axis)
          Z: inherited from argument(s)
          T: inherited from argument(s)
    DAT_TO_SAMPLE: variable (x,y,z,t) to sample
        Influence on output grid:
          X: no influence (indicate argument limits with "[]")
          Y: no influence (indicate argument limits with "[]")
          Z: passed to result grid
          T: passed to result grid
    XPTS: X indices of sample points
        Influence on output grid:
          X: no influence (indicate argument limits with "[]")
          Y: no influence (indicate argument limits with "[]")
          Z: no influence (indicate argument limits with "[]")
          T: no influence (indicate argument limits with "[]")
    YPTS: Y indices of sample points
        Influence on output grid:
          X: no influence (indicate argument limits with "[]")
          Y: no influence (indicate argument limits with "[]")
          Z: no influence (indicate argument limits with "[]")
          T: no influence (indicate argument limits with "[]")
```

## Ref Sec31.7. SHOW GRID

/I/J/K/L /X/Y/Z/T /ALL /DYNAMIC

Shows the name and axis limits of a grid.

```
yes? SHOW GRID[/qualifiers]   [var_or_grid1 var_or_grid2 ...]
```

**Example:**

(See command SHOW AXIS, p. 317, for an explanation of the columns.)

```
yes? SET DATA levitus_climatology
yes? SHOW GRID salt
    GRID GLEVITR1
 name       axis           # pts   start                end
 XAXLEVITR LONGITUDE       360mr   20.5E                19.5E(379.5)
 YAXLEVITR LATITUDE        180 r   89.5S                89.5N
 ZAXLEVITR DEPTH(-)         20 i   0m                   5000m
```

**Parameters**

The parameter(s) may be the name of one or more grid(s) or variable(s). If no parameter is given SHOW GRID displays the grid of the last variable accessed. This is the only mechanism to display the grid of an algebraic expression.

**Note:** To apply SHOW GRID to an algebraic expression it is necessary for Ferret to have evaluated the expression in a previous command. The command LOAD is useful for this purpose in some circumstances.

**Command qualifiers for SHOW GRID:**

SHOW GRID/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Displays the coordinates and grid box sizes for the specified axis. Optionally, low and high limits and a delta value may be specified to restrict the range of values displayed.

```
yes? SHOW GRID/X[=lo:hi:delta] [variable_or_grid]
```

**Example:**

```
yes? SHOW GRID/L=1:12:3  sst[coads_climatology]
```

SHOW GRID/**ALL**
Shows the names only of all grids defined.

```
yes? SHOW GRID/ALL
```

SHOW GRID/**DYNAMIC**
Shows the names of dynamic grids that are defined.

```
yes? SHOW GRID/DYNAMIC
```

## Ref Sec31.8. SHOW LIST

Shows the current states of the LIST command.

```
yes? SHOW LIST
```

The qualifier /ALL may be used with this command but exists merely for compatibility reasons and has no effect.

## Ref Sec31.9. SHOW MEMORY

/ALL/FREE/PERMANENT/TEMPORARY

Shows the state of the memory cache.

```
yes? SHOW MEMORY
```

Shows the current size of the cache.

```
yes? SHOW MEMORY[/qualifiers]
```

**Command qualifiers for SHOW MEMORY:**

SHOW MEMORY/**ALL**
Shows all variables currently cached in memory—permanent and temporary.

SHOW MEMORY/**FREE**
Shows cache memory and memory table space that remains unused.

Cache memory is organized into "blocks." One block is the smallest unit that any variable stored in memory may allocate. The total number of variables that may be stored in memory cannot exceed the size of the memory table. The "largest free region" gives an indication of memory fragmentation. A typical SHOW MEMORY/FREE output looks as below:

```
total memory table slots: 150
total memory blocks: 500
memory block size:1600

number of free memory blocks: 439
largest free region: 439
number of free regions:   1
free memory table slots: 149
```

SHOW MEMORY/**PERMANENT**
Lists the variables cached in memory and cataloged as permanent. These variables will not be deleted even when memory space is needed. They become cataloged in memory as permanent when the LOAD/PERMANENT command is used.

SHOW MEMORY/**TEMPORARY**
Lists the variables cached in memory and cataloged as temporary (they may be deleted when memory capacity is needed).

## Ref Sec31.10. SHOW MODE

Shows the names, states and arguments of the Ferret SET MODE command.

```
SHOW MODE [partial_mode_name1,name2,...]
```

**Example:**

```
yes? SHOW MODE VERIFY,META
```

The qualifier /ALL may be used with this command but exists merely for compatibility reasons and has no effect.

## Ref Sec31.11. SHOW MOVIE

Shows the current state of SET MOVIE. This state affects FRAME and graphics commands specified with the /FRAME qualifier.

```
yes? SHOW MOVIE
```

The qualifier /ALL can be used with this command, but it exists for compatibility purposes only and has no effect.

## Ref Sec31.12. SHOW QUERIES

Queries are a vehicle for communication between Ferret and a stand-alone interface program. They are not supported for general use.

## Ref Sec31.13. SHOW REGION

Shows the current default region or the named region.

```
yes? SHOW REGION[/ALL]  [region_name]
```

The region displayed is formatted appropriately for the axes of the last data accessed. For example, suppose the region along the Y axis was specified as Y=5S:5N. Then if the Y axis of the last data accessed is in units of degrees-latitude the Y location is shown as Y=5S:5N but if the Y axis of the last data accessed is “ABSTRACT” then the Y location is shown as Y=-5:5.

## Ref Sec31.14. SHOW SYMBOL

/ALL

Shows the value of one or more symbols (string variables).

```
yes? SHOW SYMBOL[/qualifier]  [symbol_name]
```

If no qualifier or symbol name is given then all defined symbols are listed. SHOW SYMBOL will accept partial names such as

```
yes? SHOW SYMBOL *lab*
   MY_X_LABEL = "Sample Number"
   LABEL_2 = "Station at 23N"
```

**Parameters**

The parameter may be the name of a symbol, with * replacing part of the string as above.

**Command qualifiers for SHOW SYMBOL:**

SHOW SYMBOL/ALL
Lists all symbols that are defined.

## Ref Sec31.15. SHOW TRANSFORM

Shows the available transformations, including regridding transformations.

```
yes? SHOW TRANSFORM
```

**Note:** This is the most reliable way to view transformations. The output of this command will be current even when this manual is out of date.

The qualifier /ALL may be used with this command but exists merely for compatibility reasons and has no effect.

## Ref Sec31.16. SHOW VARIABLES

/ALL /DATASET /DIAGNOSTIC /USER

Lists diagnostic or user-defined variables.

```
SHOW VARIABLES[/qualifier]   [partial_name]
```

**Examples:**

```
yes? SHOW VARIABLES        !all user-defined variables
yes? SHOW VAR/DIAG Q       !all diagnostic vars beginning with Q
```

**Command qualifiers for SHOW VARIABLES:**

SHOW VARIABLES/**ALL**
Lists both diagnostic variables (available for the COX/PHILANDER model) and user-defined variables.

SHOW VARIABLES/**DATA_SET**
Lists variables associated with the named dataset by DEFINE VARIABLE/DATA_SET=

SHOW VARIABLES/**DIAGNOSTIC**
This is an unsupported (obsolete) qualifier. It lists "diagnostic" variables available for the COX/PHILANDER model.

SHOW VARIABLES/**USER**
Lists expressions that have been defined by the user as new variables. This is the default behavior of SHOW VARIABLES with no qualifier.

## Ref Sec31.17. SHOW VIEWPORT

Shows one or more of the currently defined viewports. Omitting an argument gives information on all viewports.

```
yes? SHOW VIEWPORT [view_name1,view_name2,...]
```

The qualifier /ALL may be used with this command but exists merely for compatibility reasons and has no effect.

## Ref Sec31.18. SHOW WINDOWS

Lists open window numbers and indicates which is the active one.

```
yes? SHOW WINDOWS
```

The qualifier /ALL may be used with this command but exists merely for compatibility reasons and has no effect.

## Ref Sec32. SPAWN

Executes a command line (Unix shell) command from within Ferret.

```
yes? SPAWN unix_shell_command
```

**Example:**

```
yes? SPAWN rm -f file.dat
```

Also, "SPAWN shell_name" allows the user to fork into an interactive shell. For example:

```
yes? SPAWN csh
```

enters the user into a c-shell. Use EXIT to return to Ferret.

## Ref Sec33. STATISTICS

/I/J/K/L X/Y/Z/T /D /BRIEF

Computes summary statistics about the data specified.

```
yes? STATISTICS[/qualifiers]    expression_1 , expression_2 , ...
```

The statistics include:

- the size and shape of the region
- total number of data values in the region specified
- number of data values flagged as bad data
- minimum value
- maximum value
- mean value (arithmetic mean—not weighted by grid spacing)
- standard deviation (also not weighted by grid spacing)

All values are reported to 5 significant digits.

STATISTICS interacts with the current context exactly as the commands CONTOUR, PLOT and LIST do.

**Parameters**

Expressions may be anything described under Expressions. If multiple variables or expressions are specified they are treated in sequence. The expression(s) are inferred from the current context if omitted from the command line.

**Command qualifiers for STATISTICS:**

STATISTICS/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when computing statistics about the expression(s).

STATISTICS/**D=**
Specifies the default data set to be used when computing statistics about the expression(s).

STATISTICS/**BRIEF**
Produces a shorter listing involving less computation.

## Ref Sec34. UNALIAS

Alias for CANCEL ALIAS.

## Ref Sec35. USE

The USE command is an alias for SET DATA/FORMAT=cdf. (p. 290)

All qualifiers and restrictions are identical to SET DATA/FORMAT=cdf. If no filename extension is given, ".cdf" is assumed.

**Example:**

```
yes? USE test
```

is equivalent to

```
yes? SET DATA/FORMAT=cdf test
```

## Ref Sec36. USER

Executes a user-written extension to the Ferret program.

```
yes? USER[/COMMAND=]    expression_1 , expression_2, ...
```

The USER command is a means of incorporating custom changes in Ferret. It is currently supported only by special request to the Ferret developers (ferret@pmel.noaa.gov). Two special features are currently accessible through the USER command—objective analysis and scattered sampling of grids. These commands are superceded with Version 5.0 by the functionality available through the external functions.

We recommend the user access objective analysis via the script objective.jnl. The scattered sampling feature is used in the polar plotting GO tools (try "GO polar_demo" at the Ferret prompt).

## Ref Sec36.1. Objective analysis

(**Note:** see the version 4.4 documentation for an older way of gridding (X,Y, value) triples onto a grid)

To grid a set of (X, Y, value) triples onto a grid of specified resolution, sometimes called Objective analysis, use one of the family of "scat2grid" external functions. See the description of these functions in Appendix A starting at p. 68.

```
yes? SHOW FUNCTION/EXTERNAL scatter*
SCAT2GRIDGAUSS_XY(X,Y,Z,XAX,YAX,CUTOFF)
     Use Gaussian weighting to grid scattered data to an XY grid.
     X: X coordinates of scattered input triples
     Y: Y coordinates of scattered input triples
     Z: Z = F(X,Y) Z component of scattered input triples
     XAX: X axis of output grid
```

```
     YAX: Y axis of output grid
      CUTOFF: Interpolation parameter: cutoff limit
SCAT2GRIDGAUSS_XZ(X,Z,F,XAX,ZAX,CUTOFF)
...
```

The X, Y, and F(X,Y) are lists of locations and a value associated with each location. Define X and Y axes of the desired the grid and call the function to interpolate these points to the grid. Say you have a set of latitudes, longitudes, and samples of a quantity N03 at those points, and that these are in the variables my_lat, my_lon, and n03.

```
yes? DEFINE AXIS/X=170W:120W:5 xax5
yes? DEFINE AXIS/Y=0:40N:5 yax5
yes? LET n03_reg = scat2gridgauss_xy(my_lat, my_lon, n03, xax5, yax5,
2.)
yes? SHADE n03_reg
```

See also the example in the demo script,

```
yes? go objective_analysis_demo
```

## Ref Sec36.2. Scattered sampling

**Note:** there was an older way of doing scattered sampling; see section 33.2 in the version 4.4 documentation)

Ferret  functions are available for sampling a gridded data field. See

```
yes? SHOW FUNCTION sample*

SAMPLEI(DAT_TO_SAMPLE,I_INDICES)
SAMPLEJ(DAT_TO_SAMPLE,J_INDICES)  ! These sample a gridded field,
returning
SAMPLEK(DAT_TO_SAMPLE,K_INDICES)  ! data at a set of grid points along
an
SAMPLEL(DAT_TO_SAMPLE,L_INDICES)  ! axis

SAMPLEIJ(DAT_TO_SAMPLE,XPTS,YPTS) ! Returns data sampled at a
2-dimensional
                                  ! subset of its grid points

SAMPLET_DATE(DAT_TO_SAMPLE,YR,MO,DAY,HR,MIN,SEC)  ! Returns data
sampled by
                                  ! interpolating to one or more times

SAMPLEXY(DAT_TO_SAMPLE,XPTS,YPTS) ! Returns data sampled at a set of
(X,Y)
                                  ! points, i.e., a ship track or some
                                  ! other path, using linear
interpolation
```

Examples of the use of some of these functions are in ef_sort_demo.jnl

# Ref Sec37. VECTOR

/I/J/K/L /X/Y/Z/T /D /ASPECT /FRAME /LENGTH /NOAXIS /NOLABELS /OVERLAY /PEN /SET_UP /TITLE /COLOR /TRANSPOSE /XLIMITS /XSKIP /YLIMITS /YSKIP

Produces a vector arrow plot.

```
VECTOR[/qualifiers] x_expr,y_expr
```

**Parameters**

x_expr, y_expr
Algebraic expressions (or simple variables) specifying the x components and y components of the vector arrows. The expression pair will be inferred from the current context if omitted from the command line.

**Command qualifiers for VECTOR:**

VECTOR/**I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression being plotted.

VECTOR/**D=**
Specifies the default data set to be used when evaluating the expression pair being plotted.

VECTOR/**ASPECT**
Adjusts the direction of the vectors to compensate for differing axis scaling.

```
yes? VECTOR/ASPECT[=aspect_ratio]  x_expr, y_expr...
```

The size of vectors is unchanged—only the direction is modified. Under most circumstances /ASPECT should be specified. The aspect ratio is (Y-scale/X-scale). If the plot lies in the latitude/longitude plane the aspect ratio correction will be adjusted as a function of COS(LATITUDE) on the plot.

For example, in a typical oceanographic XZ plane plot the vertical (Z) axis is in tens of meters while the horizontal (X) axis is in hundreds of kilometers. This means the vertical scale is greatly magnified in comparison to the horizontal. The /ASPECT qualifier correspondingly magnifies the vertical component of the vector relative to the horizontal while preserving the length of the vector. The magnification factor is documented on the plot.

If no aspect ratio is specified by the user (e.g. VECTOR/ASP with no value), then Ferret will plot the vectors so that the two components' relative sizes shows their ratio. (In an XZ plane, then, ocean velocity vectors will nearly always appear horizontal) .

VECTOR/**FRAME**
Causes the graphic image produced to be captured as an animation frame in the file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

VECTOR/**LENGTH=**
Controls the size of vectors.

```
yes? VECTOR/LENGTH[=value_of_standard]
```

If the /LENGTH qualifier is omitted Ferret automatically selects reasonable vector lengths. To reuse the vector length from the last VECTOR plot use VECTOR/LENGTH.

To specify the vector lengths manually use the value_of_standard argument. This associates the value "val" with the standard vector length, normally ½ inch. Note that the PPLUS command VECSET can be used to modify the length of the standard vector. This is also the length that is displayed in the vector key.

**Example:**

```
yes? VECTOR/LENGTH=100 U,V
```

Creates a vector arrow plot of velocities with ½ inch vectors for speeds of 100.

VECTOR/**NOAXIS**
Suppresses all axis lines, tics, and labeling so that no box appears surrounding the contour plot. This is especially useful for map projection plots.

VECTOR/**NOLABELS**
Suppresses all plot labels except axis labels.

VECTOR/**OVERLAY**
Causes the indicated vector field to be overlaid on the existing plot.

VECTOR/**COLOR=**
Specifies the line color for the vectors. The available color names are Black, Red, Green, Blue, LightBlue, Purple, and White (not case sensitive), corresponding to the /PEN values 1-6, respectively. (/COLOR also accepts numerical values.). Note that White is only available for the default THICKNESS=1.

```
yes? VECTOR/PEN=green   x_expr, y_expr
```

VECTOR/**PEN=**
Specifies the line style for the vectors. /PEN= takes the same arguments as the /LINE= qualifier for command PLOT. See command PLOT/LINE= (p. 280). "n" ranges from 1 to 18.

```
yes? VECTOR/PEN=n   x_expr, y_expr
```

VECTOR/**SET_UP**
Performs all the internal preparations required by program Ferret for vector plots but does not actually render output. The command PPL can then be used to make changes to the plot prior to producing output with the PPL VECTOR command. This permits plot customizations that are not possible with Ferret command qualifiers. See the chapter "Customizing Plots" (p. 129).

VECTOR/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about x_expr and y_expr. To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character. See the chapter "Customizing Plots", section “Fonts” (p. 149).

```
yes? VECTOR/TITLE="title_string"  x_expr, y_expr
```

VECTOR/**TRANSPOSE**
Causes the horizontal and vertical axes to be interchanged. By default the X axis is always drawn horizontal and the Y and Z axes are drawn vertical. For Y-Z plots the Z data axis is vertical.

VECTOR/**XLIMITS=**
Specifies X axis limits and tic interval. Without this qualifier, Ferret selects reasonable values.

```
yes? VECTOR/XLIMITS=lo:hi:increment   x_expr, y_expr
```

The optional “increment” parameter determines tic mark spacing on the axis. If the increment is negative, the axis will be reversed.

VECTOR/**XSKIP=/YSKIP=**
Draws every nth vector along the requested axis beginning with the first vector requested.

```
yes? VECTOR/XSKIP=nx/YSKIP=ny  u,v
```

By default, Ferret “thins” vectors to achieve a clear plot. These qualifiers allow control over thinning.

Note that when the /SETUP qualifier is used the /XSKIP and /YSKIP qualifiers are ignored. In this case, use arguments to the PPL VECTOR command to achieve the thinning.

```
PPL VECTOR/qualifiers,xskip,yskip
```

VECTOR/**YLIMITS=**
Specifies Y axis limits and tic interval. See /XLIMITS= (above).

## Ref Sec38. WHERE

The command (alias) WHERE requests mouse input from the user, using the indicated click position to define the symbols XMOUSE and YMOUSE in units of the plotted data. Comments that include the digitized position are also written to the current journal file (if open). The WHERE command can be embedded into scripts to allow interactive positioning of color keys, boxes, lines, and other annotations.

## Ref Sec39. WIRE

/I/J/K/L /X/Y/Z/T /D /FRAME /NOLABEL /OVERLAY/SET_UP /TITLE /TRANSPOSE /VIEWPOINT /ZLIMITS /ZSCALE

Produces a wire frame representation of a two-dimensional field.

```
yes? WIRE[/qualifiers] expression
```

**Parameters**

The expression may be anything described in the chapter "Variables and Expressions", section “Expressions” (p. 53). The expression will be inferred from the current context if omitted from the command line. Multiple expressions are not permitted in a single WIRE command. The indicated region should denote a plane (2D) of data.

**Command qualifiers for WIRE:**

WIRE**/I=/J=/K=/L=/X=/Y=/Z=/T=**
Specifies value or range of axis subscripts (I, J, K, or L), or axis coordinates (X, Y, Z, or T) to be used when evaluating the expression being plotted.

**Example:**

The following commands will create a wire frame representation of a simple mathematical function in two dimensions.

```
yes? SET REGION/I=1:80/J=1:80
yes? WIRE/VIEWPOINT=-4,-10,2 exp(-1*(((I-40)/20)^2 + ((J-40)/20)^2))
```

WIRE**/D=**
Specifies the default data set to be used when evaluating the expression being plotted.

WIRE**/FRAME**
Causes the graphic image produced to be captured as an animation frame in the file specified by SET MOVIE. In general the FRAME command (p. 265) is more flexible and we recommend its use rather than this qualifier.

WIRE/**NOLABEL**
Suppresses all plot labels except axis labels.

WIRE/**OVERLAY**
Causes the indicated wire frame plot to be overlaid on the existing plot.

WIRE/**SET_UP**
Performs all the internal preparations required by program Ferret for wire frame graphics but does not actually render output. The command PPL can then be used to make changes to the plot prior to producing output with the PPL WIRE command. This permits plot customizations that are not possible with Ferret command qualifiers. See the chapter "Customizing Plots" (p. 129).

WIRE/**TITLE=**
Allows user to specify a plot title (enclosed in quotation marks). Without this qualifier Ferret selects a title based on information about the expression. To include font change and color_thickness specifications (e.g., @TI@C002) in the title string, it is necessary either to CANCEL MODE ASCII or to include a leading ESC (escape) character. See the chapter "Customizing Plots", section “Fonts” (p. 149).

WIRE/**TRANSPOSE**
Causes the X and Y axes to be interchanged.

WIRE/**VIEWPOINT=**
Specifies a viewpoint for viewing the wire frame.

```
yes? WIRE/VIEWPOINT=x,y,z   expression
```

The x,y values are specified as coordinates on the X and Y axes (though they may exceed the axis limits). The z value is in units of the requested variable.

WIRE/**ZLIMITS=**
Specifies limits of Z axis for wire frame.

```
yes? WIRE/ZLIMITS=zmin,zmax,delta   expression
```

The values given are in units of the requested variable. (The string given as an argument to /ZLIMITS= is passed unmodified to the PPLUS command WIRE as the zmin and zmax parameters.)

WIRE/**ZSCALE=**
Controls Z axis scaling of the 3-D plot.

```
yes? WIRE/ZSCALE=s  expression
```

The default value is equivalent to (ymax-ymin)/(zmax-zmin) (i.e., the aspect ratio of the Z axis to the Y axis). This qualifier is identical to the PPLUS VIEW command parameter of the same name.

# GLOSSARY

**ABSTRACT EXPRESSION (or VARIABLE)**
An expression which contains no dependencies on any disk-resident data is referred to as "abstract". For example, SIN(x), where x is a pseudo-variable.

**AXIS**
A line along one of the dimensions of a grid. The line is divided into n points, or more precisely, n grid boxes where each grid box is a length along the axis. Adjacent grid boxes must touch (no gaps along the axis) but need not be uniform in size (points may be unequally spaced). Axes may be oriented (e.g. latitude, depth, ...) or simply abstract values.

**COARDS**
A profile for the standardization of NetCDF files.

**CONTEXT**
The information needed to obtain values for a variable: the location in space and time (points or ranges), the name of the data set (if a file variable) and an optional grid.

**DATA SET**
A collection of variables in one or more disk files that may be specified with a single SET DATA command.

**DESCRIPTOR**
A file containing background data about a GT or TS-formatted data set: variable names, coordinates, units and pointers to the data files. Descriptor file names normally end with ".DES".

**DYNAMIC AXIS**
An axis that is inferred through the use of lo:hi:delta notation. It is created and destroyed dynamically by Ferret.

**DYNAMIC GRID**
A grid whose axes are inferred from a regridding operation that does not explicitly specify all of the destination axes or specifies a destination grid that can be rendered conformable with the originating grid only if some axes are removed or substituted.

**EXPRESSION**
Any valid combination of operators, functions, transformations, variables and pseudo-variables is an expression. For example, "ABS(U)", "TEMP/(-0.03^Z)" or "COS(TEMP[Y=0:40N@LOC:15])".

**EZ DATA SET**
Any disk data file that is readable by Ferret but is not in GT, TS, or NetCDF format.

**FILE VARIABLE**
A variable made available with the SET DATA command. File variables are data in disk files suitable for plotting, listing, using in user-variable definitions, etc.

**GKS**
The "Graphical Kernel System" — a graphics programming interface that facilitates the development of device-independent graphics code.

**GO FILE** or **GO SCRIPT**
A file of Ferret commands intended to be executed as a single command with the GO command.

**GRID**
A group of 1 to 4 axes defining a coordinate space. A grid can associate the axes as "outer products" creating a rectangular array of points. Grids may be defined with the DEFINE GRID command or from inside data sets.

**GRID BOX**
A length along an axis assumed to belong to a single grid point. It is represented by a box "middle", a box upper and a box lower limit. The "middle" need not actually be at the center of the box but the upper limit of box m must always be the lower limit of box m+1. (This concept is needed for integration of variables along an axis.)

**GRID FILE**
A file containing the definition of grids and axes — part of the GT and TS formats.

**GT FORMAT**
"grids at time steps" format. A direct access format using a separate descriptor file for descriptive metadata.

**METAFILE**
A representation of graphics stored in a computer file. Such a file can be processed by an interpreter program (such as Fprint) and sent to a graphics output device.

**MODULO AXIS**
An axis where the first point of the axis logically follows the last. Examples of this are degrees of longitude or dates in a climatological year.

**MODULO REGRIDDING**
A regridding operation where the destination axis is modulo and the regridding transform is a modulo operation. Typical usage would be to create a 12-month climatology from a multi-year time series.

**NETCDF**
Network Common Data Format is an interface to a library of data access routines for storing and retrieving scientific data. NetCDF allows the creation of data sets which are self-de-

scribing and network transparent. As of Ferret version 2.30, NetCDF is the suggested method of data storage.

**OPERATOR**
A function that is syntactically expressed in-line instead of as a name followed by arguments. The Ferret operators are +, -, *, /, ^, AND, OR, EQ, NE, LT, LE, GT and GE.

**PSEUDO-VARIABLE**
A special variable whose values are coordinates or coordinate information about a grid. X, I, and XBOX are the pseudo-variables for the X axis — similarly for the other axes.

**QUALIFIER**
Commands and variable names may require auxiliary information supplied by qualifiers. In the command "SHOW DATA/FULL," "/FULL" is a qualifier. In the variable "SST[Y=20N]," "Y=20N" is a qualifier.

**REGION**
The location in space and time (or other axis units) at which a variable is to be evaluated. The locations may be points or ranges. For example, T="1-JAN-1982",Y=12S:12N describes a region in latitude and time.

**REGRID**
The process of converting the values of a variable from one grid to another. By default this is done through multi-linear interpolation along all axes from the old grid to the new. Other methods are also supported.

**SUBSCRIPT**
A coordinate system for referring to grid locations in which the points along an axis are regarded as integers from 1 to the number of points on the axis. The qualifiers I, J, K, and L are provided to specify locations by subscript.

**TRANSFORMATION**
An operation performed on a variable along a particular axis and specified via the syntax "@trn". Some transformations, such as averaging (e.g. U[Z=@AVE]), reduce the range of the variable along the axis to a single point. Others, such as taking a derivative (e.g., V[T=@DDC]) do not.

**TMAP-FORMAT**
Special formats created by the Thermal Modeling and Analysis Project (TMAP). These formats use descriptor files to store information about the variables, units, titles, and grids for the data. Separate formats allow optimized access as time series (TS format) or as geographical regions (GT format). As of Ferret version 2.30, NetCDF is the suggested method of data storage.

**TS FORMAT**
"time step" format. A direct access format using a separate descriptor file for descriptive metadata.

**USER-DEFINED VARIABLE**
A variable created with DEFINE VARIABLE (alias LET).

**VARIABLE**
Value defined on a grid.

**VARIABLE NAME**
The name by which a variable will be indicated in commands and expressions. Names begin with letters and may include letters, digits, dollar signs, and underscores.

**VARIABLE TITLE**
A title string used to label plots and listed outputs of a variable.

**VIEWPORT**
A graphical display region which may be any subrectangle of a window. Graphical commands (PLOT, CONTOUR, etc.) take complete control of a viewport, clearing it as needed. A window may contain several viewports — possibly overlapping. Viewports are defined with DEFINE VIEWPORT and controlled with SET and CANCEL VIEWPORT.

**WINDOW**
A rectangular graphical display region. On a graphics terminal the terminal screen is the one and only window available. On a graphics workstation there may be many output windows.

**WORLD COORDINATE**
A coordinate system for referring to grid locations in which the points along an axis are regarded as continuous values in some particular units (e.g., meters of depth, degrees of latitude). The qualifiers X, Y, Z, and T are provided to specify locations by world coordinate.

# APPENDIX: EXTERNAL FUNCTIONS

A number of external functions are included with the Ferret distribution. This number is expected to grow as Ferret developers and users contribute more functions. See the chapter "Writing External Functions" (p. 215) for how to convert your Fortran code to a Ferret external function. Send your contributions to the Users Guide editor at ferret_ug@pmel.noaa.gov, or the Ferret developers at ferret@pmel.noaa.gov.

The functions are listed in the following sections. To see what functions are available to you, type

```
yes? SHOW FUNCTION/EXTERNAL
```

or

```
yes? SHOW FUNCTIONS/DETAILS/EXTERNAL function_name
```

gives further details on how the arguments influence the grid for the function's result.

## Appendix A Sec1. COMPRESSI

**COMPRESSI(DAT) Returns data, compressed along the I axis: Missing points moved to the end**

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to compress in I |
| **Result Axes:** | X | ABSTRACT, same length as DAT x-axis |
| | Y | Inherited from DAT |
| | Z | Inherited from DAT |
| | T | Inherited from DAT |

**Note:**
It is generally advisable to include explicit limits when working with functions that replace axes. for example, consider the function compressi(v). The expression

```
list/i=6:10 compressi(v)
```

is not equivalent to

```
list compressi(v[i=6:10])
```

The former will list the 6th through 10th compressed indices from the entire i range of variable v. the latter will list all of the indices that result from compressing v[i=6:10].

## Appendix A Sec2. COMPRESSJ

COMPRESSJ(DAT) Returns data, compressed along the J axis: Missing points moved to the end

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to compress in J |
| **Result Axes:** | X | Inherited from DAT |
| | Y | ABSTRACT, same length as DAT y-axis |
| | Z | Inherited from DAT |
| | T | Inherited from DAT |

**Note:** see the note under COMPRESSI on specifying axis limits (p. 63)

## Appendix A Sec3. COMPRESSK

**COMPRESSK(DAT)** Returns data, compressed along the I axis: Missing points moved to the end

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to compress in K |
| **Result Axes:** | X | Inherited from DAT |
| | Y | Inherited from DAT |
| | Z | ABSTRACT, same length as DAT z-axis |
| | T | Inherited from DAT |

**Note:** see the note under COMPRESSI on specifying axis limits (p. 63)

## Appendix A Sec4. COMPRESSL(DAT)

**COMPRESSL(DAT)** Returns data, compressed along the L axis: Missing points moved to the end

| | | |
|---|---|---|
| **Arguments:** | DAT | DAT: variable to compress in L |
| **Result Axes:** | X | Inherited from DAT |
| | Y | Inherited from DAT |
| | Z | Inherited from DAT |
| | T | ABSTRACT, same length as DAT t-axis |

**Note:** see the note under COMPRESSI on specifying axis limits (p. 63)

## Appendix A Sec5. CONVOLVEI

**CONVOLVEI (VAR, WEIGHT), CONVOLVEJ (VAR, WEIGHT) , CONVOLVEK (VAR, WEIGHT), CONVOLVE L (VAR, WEIGHT)** Convolve I (J,K,or L)component of variable with weight function

| | | |
|---|---|---|
| **Arguments:** | COM | COM: variable to convolve |
| | WEIGHT | Weight function |
| **Result Axes:** | X | Inherited from COM |
| | Y | Inherited from COM |
| | Z | Inherited from COM |
| | T | Inherited from COM |

This function (and likewise CONVOLVEJ, CONVOLVEK, and CONVOLVEL) convolves the variable VAR, with the weight function, wt along the X axis. Note that the variable's context may not be of adequate size for the full calculation. Missing data flags will be inserted where computation is impossible.

When bad data points are encountered in the component data all result data depending on it are flagged as bad, too.

The weight function is applied at each point from i-hlen to i+hlen, where hlen is half the length of the weight function. If the function is of even length, a zero weight is used at the upper end. Thus if the weights were {0.1, 0.4, 0.4, 0.1} the result at point I would be computed as the sum 0.1* COM(i-2) + 0.4* com(i-1) + 0.4* COM(i) + 0.1* COM(i+1) + 0.* COM(i+2)

**Example:**

Use the function to smooth a function. (Figure A_1)

```
yes? LET weight = {0.25, 0.5, 0.25}
```

```
yes? LET c = SIN(x[x=0:10:.1]) + RANDU(X[X=0:10:.1])/5
yes? PLOT c
```

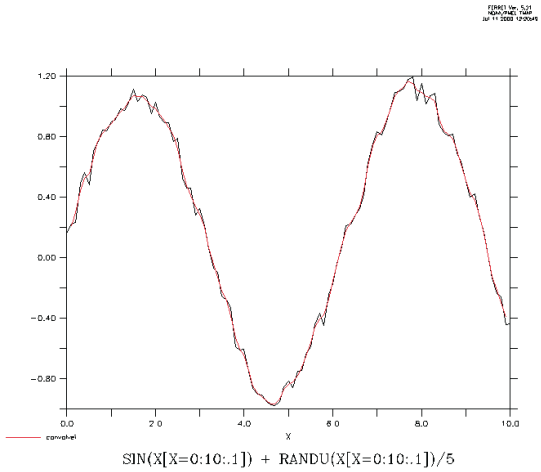



Figure A_1

```
yes? PLOT/OVER/TITLE="convolvei" CONVOLVEI(c,weight)
```

## Appendix A Sec6. EOF_SPACE

**EOF_SPACE(A,PCT_CUTOFF,FRAC_TIMESER)** Returns EOF (Empirical Orthogonal Function) spacial fields(eigenfunctions) from x-y-time field w/gaps

**Note:** The EOF functions are not fully tested but are included due to user interest. Your feedback on your experience with them will help to refine the implementation of EOF's in Ferret.

| | | |
|---|---|---|
| **Arguments:** | A | Variable in x, y, t; may be a function of z |
| | PCT_CUTOFF | Return eigenfunctions up to cutoff for % variance explained e.g. 2 for 2% |
| | FRAC_ TIMESER | Use only those time series with this fraction valid data, e.g. 0.8 to require that 80% of the data be present to use the data at a location. |
| **Result Axes:** | X | Inherited from A |
| | Y | Inherited from A |
| | Z | Inherited from A |
| | T | ABSTRACT; length from A, but meaningful results are determined by argument 2, the percent variance explained by the eigenfunctions |

The EOF functions all make the same computations, returning different portions of the results. EOF_SPACE returns the eigenfunctions, normalized so that they have the units of data, while time amplitude functions (TAF's) are dimensionless. Thus the sum of the values of a given EOF = sqrt eigenvalue), and the mean of a given TAF = 1. EOF_STAT returns some useful statistics: the number of EOF's which were computed and normalized for the parameters given; the %variation explained for each eigenfunction, and the eigenvalues.

The method is an implementation of Chelton's '82 method for finding EOFs of gappy time series. If there are no gaps, it reduces to ordinary EOFs.

See the example under EOF_STAT for more on the input parameters, and see the demonstration ef_eof_demo.jnl for examples of this function.

## Appendix A Sec7. EOF_STAT

**EOF_STAT(A,PCT_CUTOFF,FRAC_TIMESER)** Used with EOF_SPACE and/or EOF_TFUNC. Return statistics related to an EOF solution for a given set of parameters. Results are on the x-axis j = 1: # EOFscomputed and scaled, j = 2: % percentage of total variance accounted for by each eigenvector, j = 3: the eigenvalues.

**Note:** The EOF functions are not fully tested but are included due to user interest. Your feedback on your experience with them will help to refine the implementation of EOF's in Ferret.

| | | |
|---|---|---|
| **Arguments:** | A | Variable in x, y, t; may be a function of z |
| | PCT_CUTOFF | Return eigenfunctions up to cutoff for % variance explained e.g. 2 for 2% |
| | FRAC_ TIMESER | Use only those time series with this fraction valid data, e.g. 0.8 to require that 80% of the data be present to use the data at a location. |
| **Result Axes:** | X | ABSTRACT: meaningful results determined by argument 2: % variance explained by the eigenfunctions |
| | Y | ABSTRACT: 1 through 3 as outlined in the description. |
| | Z | NORMAL (no axis) |
| | T | NORMAL (no axis) |

Please see the discussion under EOF_SPACE, and see the demonstration ef_eof_demo.jnl for examples of this function.

**Example results:**

For a simple sample function, eof_stat called to decompose it into eigenfunctions. We request that the functions returned explain at least 2% of the total variance, and allow data to be used if the time series at the point has at least 50% valid data.

Request the number of eigenvalues computed for this choice of parameters.

```
yes? list/i=1/j=1 estat
             EOF_STAT(SAMPLE_FUNCTION, 2., 0.5)
             X: 1
             Y: 1
           2.000
```

Now get the percent variance explained by the eigenfunctions which were computed.

```
yes? list/i=1:5/j=2 estat
             EOF_STAT(SAMPLE_FUNCTION, 2., 0.5)
             2
             2
 1   / 1:  93.59
 2   / 2:   6.41
 3   / 3:   0.00
 4   / 4:   0.00
 5   / 5:   0.00
```

And finally the eigenvalues associated with these eigenfunctions.

```
yes? list/i=1:5/j=3 estat
             EOF_STAT(SAMPLE_FUNCTION, 2., 0.5)
             3
             3
 1   / 1:  1104.
 2   / 2:    76.
 3   / 3:     0.
 4   / 4:     0.
 5   / 5:     0.
```

## Appendix A Sec8. EOF_TFUNC

**EOF_TFUNC(A,PCT_CUTOFF,FRAC_TIMESER)**Compute EOF time amplitude functions from x-y-time field w/gaps.

**Note:** The EOF functions are not fully tested but are included due to user interest. Your feedback on your experience with them will help to refine the implementation of EOF's in Ferret.

| | | |
|---|---|---|
| **Arguments:** | A | Variable in x, y, t; may be a function of z |
| | PCT_CUTOFF | Return eigenfunctions up to cutoff for % variance explained e.g. 2 for 2% |

| | | |
|---|---|---|
| | FRAC_ TIMESER | Use only those time series with this fraction valid data, e.g. 0.8 to require that 80% of the data be present to use the data at a location. |
| **Result Axes:** | X | ABSTRACT: meaningful results determined by argument 2: % variance explained by the eigenfunctions |
| | Y | NORMAL (no axis) |
| | Z | Inherited from A |
| | T | Inherited from A |

Please see the discussion under EOF_SPACE, and see the demonstration ef_eof_demo.jnl for examples of this function.

The time amplitude functions (TAF's) are dimension less; and the mean of a given TAF = 1. They are returned as follows: For x=1, time amplitude function corresponding to the first eigenfunction is the time series with t=1:NT.

## Appendix A Sec9. WRITEV5D

**WRITEV5D(V1,V2,V3,V4,V5,V6,V7,V8,FILENAME)** Write up to 8 variables to a Vis5D-formatted file V5.1

| | | |
|---|---|---|
| **Arguments:** | V1 | |
| | V2 | |
| | V3 | Up to 8 variables to write to the file |
| | V4 | |
| | V5 | |
| | V6 | |
| | V7 | |
| | V8 | |
| | FILENAME | Name of the file to write: file type for Vis5d files is .v5d |
| **Result Axes:** | X | Inherited from variables: all variables must have the same x and y axes |
| | Y | Inherited from variables: all variables must have the same x and y axes |
| | Z | Inherited from variables; the result grid will contain the union of all the levels that are present in the variables. |
| | T | Inherited from variables: all variables must have the same time axis |

This function calls utility functions from the Vis5D distribution to write a Vis5D-formatted file containing Ferret variables. TheVis5D tool is a system for interactive visualization of large 5-D gridded data sets. It was developed by Bill Hibbard and others at the University of Wisconsin, and can be found at

http://www.ssec.wisc.edu/~billh/vis5d.html

There are limits in Vis5D on the size of the grid and the number of timesteps. The function will issue an error if these limits are exceeded.

To make it more convenient to call the writev5d function, to open Vis5D from Ferret, and to append to a Vis5D file, GO tools are available: vis5d_write.jnl, vis5d_start.jnl, andvis5d_append.jnl. These have the filename first in their argument lists, and do not require the user to specify all 8 arguments to the function.

Write 3 variables to a file, then append to some of the variables. The times in this example have a gap in them; this will show up in the Vis5d tool as a gap in time. Last, start Vis5d and open the file.

```
Yes? SET REGION/I=55:180/J=30:60
yes? GO vis5d_write tm_1.v5d sst[L=20:30], airt[L=20:30], fcn_1
yes? GO vis5d_append t_1.v5d sst[l=34,50], airt[l=34,50]
yes? GO vis5d_start temper_1.v5d
```

See the demonstration ef_wv5d_demo.jnl for examples of this function.

## Appendix A Sec10. ZAXREPLACE_AVG

**ZAXREPLACE_AVG(V,ZVALS,ZAX)** Average to interpolate V onto the Z axis of ZAX using Z values in ZVALS

| | | |
|---|---|---|
| **Arguments:** | V | A function of depth and perhaps, x, y, and time. |
| | ZVALS | |
| | ZAX | |
| **Result Axes:** | X | Inherited from V |
| | Y | Inherited from V |
| | Z | Inherited from ZAX |
| | T | Inherited from V |

## Appendix A Sec11. ZAXREPLACE_BIN

**ZAXREPLACE_BIN(V,ZVALS,ZAX)** Average within bins to put var V onto the Z axis of ZAX using Zvalues in ZVALS

| | | |
|---|---|---|
| **Arguments:** | V | A function of depth and perhaps, x, y, and time. |
| | ZVALS | |
| | ZAX | |
| **Result Axes:** | X | Inherited from V |
| | Y | Inherited from V |
| | Z | Inherited from ZAX |
| | T | Inherited from V |

# Index

## !



## A

## B

## C

# D

# E

## F

# G

# H



# I

# J



# K



# L

## M



SET MEMORY 300



SET MODE DESPERATE 304

strategies 189

## N



## O

## P

## Q



## R



## S

## T

## U



## V

## W



## X



## Y

## Z