A Users' Guide to RIP Version 4: A Program for Visualizing Mesoscale Model Output

Mark T. Stoelinga
University of Washington

March 2009

Table of Contents

1. Introduction

RIP (which stands for <u>R</u>ead/<u>I</u>nterpolate/<u>P</u>lot) is a Fortran program that invokes NCAR Graphics routines for the purpose of visualizing output from gridded meteorological data sets, primarily from mesoscale numerical models. It was originally designed for sigma-coordinate-level output from the PSU/NCAR Mesoscale Model (MM4/MM5), but was generalized in April 2003 to handle data sets with any vertical coordinate, and in particular, output from the Weather Research and Forecast (WRF) modeling system. It can also be used to visualize model input or analyses on model grids. It has been under continuous development since 1991, primarily by Mark Stoelinga at both NCAR and the University of Washington. RIP is an officially supported component of the WRF modeling system, but could potentially be used with output from any mesoscale model.

The program is designed to be portable to any UNIX system that has a Fortran 77 or Fortran 90 compiler and the NCAR Graphics library. The UNIX operating system is not necessarily a requirement, but this document is written with the assumption that you are working on a UNIX system. The author has not given a considerable amount of thought to problems that may arise on another operating system.

RIP can be described as "quasi-interactive". You specify the desired plots by editing a formatted text file. The program is executed, and an NCAR Graphics output file is created. The plots can be modified, or new plots created, by changing the user input file and re-executing RIP. Some of the basic features of the program are outlined below:

- uses a preprocessing program (called RIPDP) which reads model output, and converts this data into standard RIP format data files that can be ingested by RIP. Three versions of RIPDP are provided: one for output from the MM5 Modeling System, one for output from the Advanced Research WRF (ARW) version of the WRF model, and one for output from the Nonhydrostatic Mesoscale Model (NMM) version of the WRF model. Other versions of RIPDP could potentially be constructed to ingest output from other mesoscale models.
- makes Lambert Conformal, Polar Stereographic, Mercator, or stretched-rotated-cylindrical-equidistant (SRCE) map backgrounds, with any standard parallels.
- makes horizontal plots of contours, color-filled contours, vectors, streamlines, or characters.
- makes horizontal plots on model vertical levels, as well as on pressure, height, potential temperature, equivalent potential temperature, or potential vorticity surfaces.
- makes vertical cross sections of contours, color-filled contours, full vectors, or horizontal wind vectors.
- makes vertical cross sections using vertical level index, pressure, log pressure, Exner function, height, potential temperature, equivalent potential temperature, or potential vorticity as the vertical coordinate.
- makes skew-T/log p soundings at points specified as grid coordinates, lat/lon coordinates, or station locations, with options to plot a hodograph or print sounding-derived quantities.
- calculates backward or forward trajectories, including hydrometeor trajectories, and calculates diagnostic quantities along trajectories.
- plots trajectories in plan view or vertical cross sections.
- makes a data set for use in the Vis5D visualization software package.
- allows for complete user control over the appearance (color, line style, labeling, etc.) of all aspects of the plots.

2. Setting up RIP on your system

a. Obtaining and unpacking the code and related files

The most recent version of the code can be downloaded from NCAR's [WRF ARW download web page](registration required). Download the compressed tar file, RIP4.TAR.gz, to your local machine, and uncompress the file (i.e., `gunzip RIP4.TAR.gz`). The tar file contains a top-level directory called RIP, in which all the RIP-related files reside. Unpacking the tar file (i.e., running `tar xf RIP4.TAR`) will create a RIP directory, as a subdirectory of your current working directory, and will place all the RIP-related files in that new RIP directory. Thus, before unpacking the RIP tar file, you should first change to a directory that you want to be the parent directory of your RIP directory. This would commonly be your home directory. The tar file contains the following directories and files:

- *CHANGES*, a text file that logs changes to the RIP tar file.
- *Doc/*, a directory that contains documentation of RIP, most notably the HTML version of the Users' Guide that you are now reading (*ripug.htm*).
- *README*, a text file containing basic information on running RIP.
- *arch/*, directory containing default compiler flags for various machines.
- *color.tbl*, a file that contains a table defining the colors you want to have available for RIP plots.
- *clean*, script to clean compiled code.
- *compile*, script to compile the code.
- *configure*, script to configure compilation flags for your system.
- *eta_micro_lookup.dat*, a file that contains "look-up" table data for the Ferrier microphysics scheme.
- *psadilookup.dat*, a file that contains "look-up" table data for obtaining temperature on a pseudoadiabat.
- *sample_infiles/*, a directory that contains sample user input files for RIP and related programs. As of this version of the User' Guide, these files include *bwave.in, grav2d_x.in, hill2d.in, qss.in, rip_sample.in, rip_wrfnmm_boston_d01.in, rip_wrfnmm_boston_d02.in, ripdp_wrfarw_sample.in, ripdp_wrfnmm_boston_d01.in, ripdp_wrfnmm_boston_d02.in, ripdp_wrfnmm_sample.in, seabreeze2d.in, sqx.in, sqy.in, tabdiag_sample.in,* and *tserstn.dat.*
- *src/*, a directory that contains all of the source code files for RIP, RIPDP, and several other utility programs. Most of these are Fortran 77 source code files with extension *.f.* In addition, there are:
    - a few *.h* and *.c* files, which are C source code files.
    - *comconst, commptf, comvctran,* and *CMASSI.comm*, which are include files that contain common blocks that are used in several routines in RIP.
    - *pointers*, an include file that contains pointer declarations used in some of the routines in RIP.
    - *Makefile*, a secondary make file used to compile and link RIP.
- *stationlist*, a file containing observing station location information.

b. The RIP_ROOT environment variable

An important environment variable for the RIP system is RIP_ROOT. RIP_ROOT should be assigned the path name of the directory where all your RIP program and utility files (*color.tbl*, *stationlist*, lookup tables, etc.) reside. If you unpacked the RIP tar file in your home directory, then the natural choice for your RIP_ROOT directory is the RIP subdirectory that was created when you unpacked the RIP tar file.

For simplicity, you should define RIP_ROOT in one of your UNIX start-up files. For example, if you use c-shell, and you have unpacked your RIP files into a directory called */users/johndoe/RIP*, and that is where you intend to have the code and utility files reside, you should include in your *.login* or *.cshrc* file the line

```
setenv RIP_ROOT /users/johndoe/RIP
```

RIP uses the system call *getenv* to retrieve the value of RIP_ROOT from within the program. If your system's Fortran does not support the system call *getenv*, you can use instead the variable *rip_root* in the *&userin* namelist in the RIP user input file (UIF) to tell RIP where to find the utility files. This is described in more detail in Chapter 4 of the Users' Guide.

In general, the utility files mentioned above require no editing or user input. The exceptions to this are the color table file, *color.tbl*, which allows you to define colors that will be used by RIP, and the observing station listing file, *stationlist*, which provides location information for observing stations that may be referenced in various aspects of the plot request. These are described below.

c. Changing the color table file

As you become more familiar with how RIP uses colors, you may want to customize *color.tbl* to suit your own preferences. In the color table, colors are defined according to fractions of red, green, and blue, with each fraction being a number from 0.0 to 1.0. Each color is given a name. You can define up to 256 different colors, but RIP will warn you if you define more than 200, because RIP also needs to make use of the 256 available color slots to define color shades for color-filled contour plots. The basic structure of the table is as follows. The first four lines are ignored – they are simply a banner that says that this is the color table. The first two colors should always be given the names *def.background* and *def.foreground*, in that order. However, their color fractions can be changed if desired. *def.background* is the color used by NCAR Graphics for any regions where no plotting instructions have been given (i.e. the default background color). *def.foreground* is the color used by NCAR Graphics for any plotting instructions in which you did not explicitly ask for a particular color (i.e. the default foreground color). Usually these are either black and white, or white and black, but the fractions can be changed to make them different colors, as already mentioned above. It is recommended that, regardless of the colors you use for *def.foreground* and *def.background*, you also have those colors defined with their appropriate names somewhere else in the table. For example, even if color

indices 0 and 1 are given the fractions 0.0,0.0,0.0 and 1.0,1.0,1.0, there should also be colors named black and white with the same fractions somewhere else in the table. The explicitly named black or white should be used when you want things to be black or white. That way, if you change *def.foreground* or *def.background* to something else, the things that should remain black and white won't change color. The table should end with a line that contains several consecutive minus signs.

To give the user added flexibility, and for compatibility with previous versions of RIP, the user can place a color table (like the one that appears in *color.tbl*) directly into a RIP user input file (UIF). This allows the user to define different color tables that are unique to a particular UIF. The color table should be placed between the namelist section and the plot specification table in the UIF. If you run RIP with a UIF that contains a color table, RIP will recognize that color table and read it instead of (NOT in addition to) the color table in your RIP_ROOT directory.

Finally, there are a few important things that you should be aware of if you will be translating your CGM plots to postscript:

- Postscript has no concept of "background color". Hence, regardless of what your index 0 color is, translation to postscript (either mono or color) will result in a blank or white background. If this is a problem, it can be circumvented by using the *-simulatebg* option on the *ctrans* command line.
- If you translate to mono postscript, all plotting instructions, regardless of the color used (black, white, red, blue, etc.), will come out black. This means any filled areas, regardless of color or color index used, will waste large amounts of toner when printed out.
- If you translate to color postscript, some monochromatic printers have the nice feature that they convert all colors to halftone gray shades.

d. Changing or adding station listings in the stationlist file

Any number of additional stations can be added to the station listing. The important thing is to keep the station information properly lined up in the appropriate columns. The important pieces of information to enter are the verbal description of the location (under column "Location"), the ICAO four-letter code identifying the station (under column "ICAO"), the latitude and longitude (under columns "Lat" and "Lon"), and the WMO number of the station (under column "WMO#"). The other information is not read by RIP. Elevation should be entered if you have it, because it may be used in the future. The author is unsure of the meaning of the information in the third column. If the WMO number is unknown or the station has no WMO number, enter "00000" in that column.

e. Compiling RIP and associated programs

Before compiling the code, make sure the environment NETCDF is set correctly to the location where your netCDF libraries are installed. Typically (*for csh shell*):
```
setenv NETCDF /usr/local/netcdf
```

Then configure compiler options for your machine by typing:
```
./configure
```

You should see a list of options for your computer (*below is an example for a Linux machine*):

```
Will use NETCDF in dir: /usr/local/netcdf
-------------------------------------------------------------
-------------
Please select from among the following supported platforms.

1. PC Linux i486 i586 i686 x86_64, PGI compiler
2. PC Linux i486 i586 i686 x86_64, Intel compiler

Enter selection [1-2] :
```

Make sure the netCDF path is correct.
Pick compile options for your machine *(if your compiler is not listed, pick one close to yours and modify configure.rip to reflect your compiler)*.

Now compile the code by typing:
```
./compile >& make.out &
```

The redirecting to the file *make.out* is optional. A successful compilation will probably take several minutes.

If the compiling of RIP fails, it is likely that the *configure.rip* file needs to be customized for your particular platform. This involves editing *configure.rip* so that the compile and link commands and options are correctly configured for the particular system you are running on. Regardless of how you accomplish the linking with NCAR Graphics, it should be pointed out that the author has found some minor problems with RIP that occur if you use a version of NCAR Graphics that is older than 4.0. Therefore, you should try to link with NCAR Graphics 4.0 (or later) if you have it. It should also be noted that the RANGS/GSHHS high-resolution map background capability is only functional if you have NCAR Graphics 4.3 (or later). RANGS/GSHHS is not the most commonly used map background outline set, so this does not represent a significant loss of functionality if you cannot use it.

The two versions of RIPDP for the WRF modeling system, *ripdp_wrfarw* and *ripdp_wrfnmm*, need to link to the NetCDF library. This library of routines is used for I/O with files in the "Network Common Data Fromat" or NetCDF, which WRF uses. The only thing the user needs to be concerned about is setting the NETCDF environment variable correct before compiling.

A successful compilation will result in the creation of several object and executable files in the *RIP/src/* directory. The make file is also set up to create symbolic links in the *RIP/*

directory to the actual executables in the *RIP/src/* directory. In UNIX, the best way to make the RIP program and other associated executables accessible to you, regardless of your current working directory, is to add the path name of your RIP directory to your executable path list (e.g., the *path* shell variable in c-shell). For example, if the above-mentioned symbolic links to the RIP executables reside in the directory *~johndoe/RIP*, and you use the c-shell in UNIX, then you should add the following line in your .cshrc file:

```
set path = ($path ~johndoe/RIP)
```

With the RIP directory in your executable path, the RIP program (and other associated utilities like *ripdp*) can be accessed easily without having to put the executables in your working directory or having to specify full path names for the executables.

f. Notes on nonstandard Fortran 77

In general, the author has attempted to adhere to Fortran 77 standards as much as possible in writing the code. However, a few nonstandard features have been included, which are still available in most f77 and f90 compiling systems.

- Namelists: For some I/O, RIP and RIPDP use a nonstandard feature known as a namelist. A namelist is a Fortran input file structure that is not officially standard to Fortran 77, but which is available in most Fortran 77 compilers. Namelists are placed in text files that are opened and read by the Fortran program. The basic structure of a namelist is, on the first line, the name of the namelist (preceded by an ampersand), and then on subsequent lines a series of "variable = value" constructs, each separated by commas, blank spaces, or a new line. (On some machines, commas are required, so it's safest to always use them.) The final line ends the namelist should contain the single word *end* preceded by an ampersand. Each variable is an actual variable that is declared in a special *namelist* statement inside the Fortran program. The variables can be of any type. The value of a character variable should be enclosed in single quotes, although it doesn't have to be if all the characters in the string are alpha-numeric. The value of a logical variable should be ".true." or ".false.". Arrays can be assigned multiple values separated by commas (although it has come to the author's attention that with some compilers such as Sun's Fortran 90, each array element must be separately referenced with its index in parentheses). Here is a general example of a namelist:

  ```
  &samplenlist
  beatles='John','Paul','George','Ringo',
  (or with Sun f90, beatles(1)='John',beatles(2)='Paul',etc.)
  primenumbers=1,2,3,5,7,11,13, flag=.true.,
  &end
  ```

  Not all the variables that are defined as part of the namelist in the Fortran program need to be given values in the namelist file that is read. Typically, all the namelist

variables are assigned default values in the program prior to the reading of the namelist file, so that they will retain those default values if they are omitted from the namelist file.

- System calls: RIP uses the following system calls: *getenv* (for retrieving the value of a UNIX environment variable from within a program), *iargc* (for retrieving the number of command line arguments), and *getarg* (for retrieving command line arguments). These system calls are quite common, and should be available in the f77 and f90 compilers from Cray, Sun, SGI, DEC, and IBM. There are also calls (commented out in the standard release) to the system routine *flush* (for flushing the standard output buffer, so that the progress of the program can be easily monitored when standard output is being sent to a file). These were commented out because they caused problems on some compilers, but you are free to reinstate them.
- RIP has been set up in a manner that requires adjustable dimensioning of local (non-argument) arrays in subroutine *driver* and a few other routines in RIP, and in subroutine *process* in RIPDP. In other words, in subroutine *driver*, for example, several arrays that are *not* in the subroutine argument list are dimensioned with integer variables that <u>are</u> in the argument list. This is technically illegal in Fortran 77, but many compilers support it because it makes dynamic memory allocation possible. It <u>is</u> standard in Fortran 90.
- RIP uses Cray pointers. This is nonstandard Fortran 77, but is supported by most Fortran 77 and Fortran 90 compilers.

3. Preparing data with RIPDP

RIP does not ingest model output files directly. First, a preprocessing step must be executed that converts the model output data files to RIP-format data files. The primary difference between these two types of files is that model output data files typically contain all times and all variables in a single file (or a few files), whereas RIP data has each variable at each time in a separate file. The preprocessing step involves use of the program RIPDP (which stands for <u>RIP</u> <u>D</u>ata <u>P</u>reparation). RIPDP reads in a model output file (or files), and separates out each variable at each time. There are several basic variables that RIPDP expects to find, and these are written out to files with names that are chosen by RIPDP (such as *uuu*, *vvv*, *prs*, etc.). These are the variable names that RIP users are familiar with. However, RIPDP can also process unexpected variables that it encounters in model output data files, creating RIP data file names for these variables from the variable name that is stored in the model output file metadata.

When you run *compile*, it should produce executable programs called *ripdp_mm5*, *ripdp_wrfarw*, and *ripdp_wrfnmm*. Although these are three separate programs (one for use with MM5 model system, one for WRF-ARW, and one for WRF-NMM), they serve the same purpose, and will be referred to collectively as *ripdp*. Also, the two WRF versions of *ripdp* may be referred to collectively as *ripdp_wrf*, without the *arw* or *nmm* specification.

a. Running RIPDP

The program has the following usage:

```
ripdp_XXX [-n namelist file] model-data-set-name [basic|all]
data file 1 data file 2 data file 3 ...
```

In the above, the "XXX" refers to either "mm5", "wrfarw", or "wrfnmm". The argument model-data-set-name can be any string you choose, that uniquely defines this model output data set. It will be used in the file names of all the new RIP data files that are created. data-file-1, data-file-2, ... are the path names (either full or relative to the current working directory) of the model data set files, in chronological order. When the program is finished, a large number of files will have been created that will reside in the current working directory. This is the RIP data that will be accessed by RIP to produce plots. See Appendix C for a description of how these files are named and the the format of their contents.

RIPDP is flexible in that it processes any variables it encounters in the model output file, even those it is not expecting, and produces files for those variables, using the variable name provided in the model output to create the file name. Any such variables can be plotted with RIP--see the description of the *feld* keyword in Appendix A.

The "basic|all" option is available only for *ripdp_wrf*. basic causes *ripdp_wrf* to process only the basic variables that RIP requires, whereas all causes *ripdp_wrf* to process all variables encountered (as in *ripdp_mm5*). If all is specified, the *discard* variable can be used in the ripdp namelist to prevent processing of unwanted variables, as with *ripdp_mm5*. However, if basic is specified, the user can request particular other fields (besides the basic fields) to be processed by setting a *retain* variable in the ripdp namelist instead of the *discard* variable. *retain* is set in the same manner as *discard*, i.e., you assign it one or more variable names in single quotes separated by commas.

To provide more user control over the processing of data, a namelist can be specified by means of the "-n" option, with namelist-file specifying the path name of the file containing the namelist, on the *ripdp_wrf* command line. Namelists are a special type of Fortran input structure that is described in Chapter 2.

The namelist file for ripdp_wrf should contain the namelist *&userin*. Examples of a namelist input files for ripdp_wrfarw and ripdp_wrfnmm, called *ripdp_wrfarw_sample.in* and *ripdp_wrfnmm_sample.in*, respectively, are provided in the RIP tar file in the *sample_infiles* directory. The *&userin* namelist in that file is shown below, followed by a description of the variables it sets. Each variable has a default value (shown in parentheses), which is the value this variable will take if its specification is omitted from the namelist.

```
&userin
ptimes=0,-72,1, ptimeunits='h', tacc=90.,
```

```
discard='LANDMASK','H2SO4', iexpandedout=1
&end
```

- *ptimes* (real array, def.=9.0E+09), *ptimeunits* (character variable, def.='h'): *ptimes* specifies the desired times for ripdp to process. A value of 9.0E+09 indicates to RIPDP that you are using *iptimes* (see below) instead of *ptimes* to specify desired times. You can specify particular times to be processed, separated by commas. The units are determined by the value of *ptimeunits*, which can be either 'h' (for hours), 'm' (for minutes), or 's' (for seconds). You can also specify a series of times in the form *A,-B,C*, which is interpreted as "times from hour *A* to hour *B*, every *C* hours" (assuming *ptimeunits*='h'). Thus, in the example above, you want RIPDP to process times from hour 0 through hour 72, every hour (assuming they are encountered in the data). Individual times can be intermixed with series specifications. Thus, `ptimes=0,1,3,-18,3,24` would cause RIPDP to process data at hours 0,1,3,6,9,12,15,18, and 24 (assuming those times are encountered).
- *iptimes* (not shown in the above example, integer array, def.=99999999): This is an integer array that also specifies desired times for RIPDP to process, but in the form of 8-digit "mdate" times (i.e. YYMMDDHH). A value of 99999999 indicates to RIPDP that you are using *ptimes* (see above) instead of *iptimes* to specify desired times. Either one or the other of *ptimes* or *iptimes* can be used, but not both. *ptimeunits* has no effect on the meaning of the values given for *iptimes*. *iptimes* can also include series specifications. For example, `iptimes=99063012,99063018,-99070103,3,99070112` would cause RIPDP to process data at times 99063012, 99063018, 99063021, 99070100, 99070103, and 99070112.

  > Note: If you want RIPDP to simply process all times encountered, you can indicate this in one of several ways. If both *ptimes* and *iptimes* are omitted from the namelist (and thus retain their default values) or are both assigned their default values, RIPDP will process all encountered times. Or, if the first (or only) value of either *ptimes* or *iptimes* is negative, RIPDP will process all encountered times.

- *tacc* (real variable, def.=1.0): This specifies a time tolerance, in seconds, for the values assigned to *ptimes* or *iptimes*. In other words, any times encountered in the model output that are within *tacc* seconds of one of the times specified in *ptimes* or *iptimes* will be processed.
- *discard* (character array, def.=' '): This specifies the names of variables that, if encountered in the model data file, will not be processed. The default value (blank) implies that all encountered fields should be processed. Do not discard any of the basic variables, such as temperature, winds, water vapor, pressure perturbation, etc. If you do, you will run into trouble when you attempt to run RIP with your data set. It is also important to note that the names assigned to the *discard* variable are compared against the variable names in the model output metadata. Therefore, you should be careful to assign names to *discard* as they

appear in the model output metadata, NOT as they appear at the ends of the the names of the RIP data files. One thing to be particularly careful about is the use of blanks versus underscores. Variable names in the model output metadata may contain blanks. When RIP uses these names to build RIP data file names for unexpected variables, it converts blanks to underscores. For example, the MM5 output variable "SOIL T 6" at hour 12 will result in the creation of a RIP data file called something like *mycase_012.00000_SOIL_T_6*. If you want to discard "SOIL T 6", you should use *discard='SOIL T 6'*, not *discard='SOIL_T_6'*.

- *retain* (character array, def.=' '): Note: works only with *ripdp_wrf*. Similar to *discard*, but instead, this specifies the names of variables that, if encountered in the model data file, should be processed, even though the user specified basic on the *ripdp_wrf* command line. The default value (blank) implies that no fields other than the basic fields should be retained.

- *iexpandedout* (integer variable, def.=0): This integer flag is relevant only to TERRAIN or REGRID output in the MM5V3 system. If the output from one of these two programs is on an expanded domain, *iexpandedout* determines whether the expanded domain will be processed (1) or the standard (unexpanded) domain will be processed (0).

Note: With some compilers such as Sun's Fortran 90, when assigning values to an array in a namelist, each array element must be separately referenced with its index in parentheses. This would apply to the *ptimes*, *iptimes*, and *discard* arrays described above. Thus, in this situation, the above namelist would appear as

```
&userin
ptimes(1)=0,ptimes(2)=-72,ptimes(3)=1, ptimeunits='h',
tacc=90., discard(1)='LANDMASK',discard(2)='H2SO4',
iexpandedout=1
&end
```

b. Special considerations for running RIPDP for WRF-NMM

The WRF-NMM model has a unique map background and grid structure, both of which introduce special challenges for processing and plotting WRF-NMM data. The map background is what can be referred to as a "stretched rotated cylindrical equidistant", or SRCE, projection. RIPDP and RIP have been made fully capable of handling this projection. The grid staggering in WRF-NMM, however, presents more complications. The staggering pattern is known as the "E-grid", which is essentially a B-grid (as used in MM5) rotated by 45 degrees.

Because of its developmental history with the MM5 model, RIP is inextricably linked with an assumed B-grid staggering system. Therefore, the easiest way to deal with E-grid data is to make it look like B-grid data. This can be done in two ways, either of which the user can choose. In the first method, we define a B-grid in which its mass ("cross or C") points collocate with all the H and V points in the E-grid, and the B-grid's velocity ("dot or D") points are staggered in the usual B-grid way. The RIPDP-created data files retain

only the E-grid data, but then when they are ingested into RIP, the E-grid H-point data are transferred directly to overlapping B-grid cross points, and non-overlapping B-grid cross points and all dot points are interpolated from the E-grid's H and V points. This is the best way to retain as much of the exact original data as possible, but effectively doubles the number of horizontal grid points in RIP, which can be undesirable. The second method is to define a completely new B-grid that has no relation to the E-grid points, possibly (or even preferably) including a different map background, but presumably with substantial overlap between the two grids, and a horizontal resolution similar to the effective resolution of the E-grid. The E-grid data is then bilinearly interpolated to the new B-grid in RIPDP and the new B-grid data is then written out to the RIPDP output data files. With this method, the fact that the original data was on the E-grid is completely transparent to the RIP plotting program.

To specify which choice to make, and to define the new map background and grid if choice 2 is made, several additional parameters need to be defined in the *&userin* namelist. An example of a full *&userin* namelist for use with *ripdp_wrfnmm* is shown below:

```
&userin
ptimes=0,-72,1,ptimeunits='h',tacc=90.,discard='LANDMASK',
iinterp=1,dskmcib=50.,miycorsib=100,mjxcorsib=100,nprojib=1
, xlatcib=30.,xloncib=-95.,truelat1ib=30.,truelat2ib=48.,
miyib=500,mjxib=500,yicornib=46.5509,xjcornib=40.0999,dskmi
b=5.
```

The additional parameters are described below:

- *iinterp* (integer, def.=0): a switch that specifies whether you want to follow "method 1--use collocated high-density B-grid" (*iinterp=0*) or "method 2--interpolate to a new B-grid" (*iinterp=1*), as described above.
- *dskmcib* (real, def.=50.0): If *iinterp=1*, this specifies the grid spacing, in km, of the coarse domain on which the new B-grid will be based.
- *miycorsib, mjxcorsib* (integer, def.=100): If *iinterp=1*, these specify the number of grid points in the *y* and *x* directions, respectively, of the coarse domain on which the new B-grid will be based.
- *nprojib* (integer, def.=1): If *iinterp=1*, this specifies the map projection number (0: none/ideal, 1: LC, 2: PS, 3: ME, 4: SRCE) of the coarse domain on which the new B-grid will be based.
- *xlatcib, xloncib* (real, def.=45.0, -90.0): If *iinterp=1*, these specify the central latitude and longitude, respectively, for the coarse domain on which the new B-grid will be based.
- *truelat1ib, truelat2ib* (real, def.=30.0, 60.0): If *iinterp=1*, these specify the two true latitudes for the coarse domain on which the new B-grid will be based.
- *miyib, mjxib* (integer, def.=75): If *iinterp=1*, these specify the number of grid points in the *y* and *x* directions, respectively, of the fine domain on which the new B-grid will be based.

- *yicornib, xjcornib* (integer, def.=25): If *iinterp=1*, these specify the coarse domain *y* and *x* locations, respectively, of the lower left corner point of the fine domain on which the new B-grid will be based.
- *dskmib* (real, def.=25.0): If *iinterp=1*, this specifies the grid spacing, in km, of the fine domain on which the new B-grid will be based.

It should be noted that, if *iinterp=1*, grid points in the new domain that are outside the original E-grid domain will be assigned a "missing value" by RIPDP. RIP (the plotting program) handles "missing value" data inconsistently--some parts of the code are designed to deal with it gracefully, and other parts are not. Therefore, it is best to make sure that the new domain is entirely contained within the original E-grid domain. Unfortunately, there is no easy way to do this. RIPDP does not warn you when your new domain contains points outside the original E-grid. The best way to go about it is by trial and error: define an interpolation domain, run RIPDP, then plot a 2-D dot-point field such as map factor on dot points (*feld=dmap*) in color contours and see if any points do not get plotted. If any are missing, adjust the parameters for the interpolation domain and try again.

4. The RIP user input file

Once the RIP data has been created with RIPDP, the next step is to prepare the user input file (UIF) for RIP. This file is a text file, which tells RIP what plots you want and how they should be plotted. A sample UIF, called *rip_sample.in*, is provided in the RIP tar file. This sample can serve as a template for the many UIFs that you will eventually create.

A UIF is divided into two main sections. The first section specifies various general parameters about the set up of RIP, in a namelist format. The second section is the plot specification section, which is used to specify what plots will be generated.

a. The namelist section

Namelists are a special type of Fortran input structure that is described in Chapter 2. The first namelist in the UIF is called *&userin*. An example of a *&userin* namelist for RIP, which is also found in the sample UIF *rip_sample.in*, is shown below, followed by a description of the variables contains. Each variable has a default value (shown in parentheses), which is the value this variable will take if its specification is omitted from the namelist.

```
&userin
idotitle=1,titlecolor='def.foreground',
ptimes=0,6,12,
ptimeunits='h',tacc=120,timezone=-7,iusdaylightrule=1,
iinittime=1,ifcsttime=1,ivalidtime=1,inearesth=0,
flmin=.09, frmax=.92, fbmin=.10, ftmax=.85,
ntextq=0,ntextcd=0,fcoffset=0.0,idotser=0,
```

```
idescriptive=1,icgmsplit=0,maxfld=10,itrajcalc=0,imakev5d=0
&end
```

<u>Some background on the RIP frame title lines</u>: The standard title at the top of the RIP frame has up to two lines, depending on what information is requested, as described below. The first shows and an initial time. The second line shows the forecast hour, and the valid time in both UTC and local time.

- *idotitle* (integer variable, def.=1) and *title* (not shown above, character variable, def.= 'auto'): These control the first part of the first title line. If *idotitle*=1 and *title*='auto', you'll get automatically generated information on the data set name (i.e., the root part of the RIP data file names for the data set being plotted) and the UIF file name (excluding the *.in* extension). If you set *title* to something else, that string will get used for the first part of the first title line instead of the automatically generated information. If *idotitle*=0, no title will be printed.
- *titlecolor* (character variable, def.='def.foreground'): This specifies the color to use for the text in the title line(s). It should be chosen from the available colors listed in the utility file *color.tbl*, described in Chapter 2.
- *iinittime* (integer variable, def.=1): If this flag is set to 1, the initial date and time (in UTC) will be printed in one of the two title lines.
- *ifcsttime* (integer variable, def.=1): If this flag is set to 1, the forecast lead time (in hours) will be printed in one of the two title lines.
- *ivalidtime* (integer variable, def.=1): If this flag is set to 1, the valid date and time (in both UTC and local time) will be printed in one of the two title lines.
- *inearesth* (integer, def.=0): This flag allows you to have the hour portion of the initial and valid time be specified with two digits, rounded to the nearest hour, rather than the standard 4-digit HHMM specification.
- *timezone* (real variable, def: -7.0): This specifies the offset from Greenwich time (UTC) for the local time zone, and is required for the correct display of local time specification of the valid time. RIP will recognize familiar time zones of N. America (-8,-7,-6,-5) and show, for example, "MST" or "EDT". For unfamiliar time zones, it will show "LST" or "LDT".
- *iusdaylightrule* (integer variable, def.=1): This flag determines whether the U.S. daylight saving rule (*i.e., daylight saving in effect between 1st Sunday in April and last Sunday in October, up to 2006. And between the 2nd Sunday in March and the first Sunday in November, since 2007*) should be applied (1) or not (0) in displaying the local time specification of valid time.
- *ptimes* (real array, def.=9.0E+09), *ptimeunits* (character variable, def.='h'): *ptimes* specifies the desired times for RIP to plot. A value of 9.0E+09 indicates to RIP that you are using *iptimes* (see below) instead of *ptimes* to specify desired times. You can specify particular times to be plotted, separated by commas. The units are determined by the value of *ptimeunits*, which can be either 'h' (for hours), 'm' (for minutes), or 's' (for seconds). You can also specify a series of times in the form *A,-B,C*, which is interpreted as "times from hour *A* to hour *B*, every *C* hours" (assuming *ptimeunits*='h'). Individual times can be intermixed with series

specifications. Thus, `ptimes=0,1,3,-18,3,24` would cause RIP to plot data at hours 0,1,3,6,9,12,15,18, and 24 (assuming those times are available).

- *iptimes* (not shown in the above example, integer array, def.=99999999): This is an integer array that also specifies desired times for RIP to plot, but in the form of 8-digit "mdate" times (i.e. YYMMDDHH). A value of 99999999 indicates to RIP that you are using *ptimes* (see above) instead of *iptimes* to specify desired times. Either one or the other of *ptimes* or *iptimes* can be used, but not both. *ptimeunits* has no effect on the meaning of the values given for *iptimes*. *iptimes* can also include series specifications. For example,
  `iptimes=99063012,99063018,-99070103,3,99070112` would cause RIP to plot data at times 99063012, 99063018, 99063021, 99070100, 99070103, and 99070112.

    Note: If you want RIP to simply plot all times available, you can indicate this in one of several ways. If both *ptimes* and *iptimes* are omitted from the namelist (and thus retain their default values) or are both assigned their default values, RIP will plot all available times. Or, if the first (or only) value of either *ptimes* or *iptimes* is negative, RIP will plot all available times.

- *tacc* (real variable, def.=1.0): This specifies a time tolerance, in seconds, for the values assigned to *ptimes* or *iptimes*. In other words, any available times in the data that are within *tacc* seconds of one of the times specified in *ptimes* or *iptimes* will be plotted. This value should be increased if, for example, your "hourly" output is not exactly on the hour. It should be decreased if you have saved consecutive time steps and need to pin point one of those time steps.

- *flmin, frmax, fbmin*, and *ftmax* (real variables, def.=.05,.95,.10,.90, respectively): These specify the left frame limit, the right frame limit, the bottom frame limit, and the top frame limit, respectively. The size of all plots will be adjusted to be the maximum that can fit in this box. The box is defined in terms of the fractional coordinate system, i.e. all four values should be between 0.0 and 1.0

- *ntextq* (integer variable, def.=0): This is the text quality specifier (0=high; 1=medium; 2=low).

- *ntextcd* (integer variable, def.=0): This is the text font specifier [0=complex (Times); 1=duplex (Helvetica)].

    Notes on ntextq and ntextcd:

    1. *ntextcd* only applies to high quality characters.
    2. Low and medium quality characters look identical except that the medium quality zero is unslashed.
    3. Medium quality characters are drawn with polylines so their size is independent of the local translator being used. The same is not true of low quality characters, which use the GKS primitive character set, whose size can vary from one system to another.

4. Any messages that involve units will always use high quality, in order to make use of superscripting for exponents.

- *fcoffset* (real variable, def.=0.0): This is an optional parameter you can use to "tell" RIP that you consider the start of the forecast to be different from what is indicated by the forecast time recorded in the model output. It serves the same purpose as the now defunct *mdatebf*, but is specified as a relative time (in hours, + or -) rather than an absolute YYMMDDHH format, so it is more useful for automated real-time applications. A non-zero *fcoffset* causes RIP to adjust the displayed initial and forecast times to match what you consider to be the beginning of the forecast. Examples: *fcoffset*=12 means you consider hour 12 in the model output to be the beginning of the true forecast (you might use this for a run with a 12-h dynamic initialization period); *fcoffset*=-12 means you consider model output time 0 to be the true forecast hour 12 (you might use this if a high-resolution domain is initialized from the hour-12 output from a low-resolution domain).

- *idotser* (integer variable, def.=0): This flag means "do time series". By setting this flag to "1", you can print out time series of any RIP variable at any station location in the model domain. In order to do this, you must also do two other things. First, you must create a simple text file with a list of locations, one per line. Locations are specified in the same manner as cross section endpoints. See the description of keyword *crsa* in [Appendix A](#) for more information on how to specify locations. The file must be called *tserstn.dat* and should reside in your current working directory. A sample version of this file is provided in the RIP tar file. Then, you must request plots of the desired fields using the plot specification table. Any field for which *ptyp=hc* is specified will be included in the time series that is created. You can plot all the fields in one frame or in separate frames. You can request any of the RIP vertical coordinates, and multiple levels, just as you would if you were generating plots. In fact, when you run RIP, it will generate the plots asked for, but in addition, it will create an ASCII file with time series data at the requested stations for any fields that were plotted with *ptyp=hc*. The name of the ASCII file that is created will be *rip-execution-name.tser*.

- *idescriptive* (integer variable, def.=1): This flag can be set to 1 to turn on the printing of more descriptive plot titles instead of the traditional, more cryptic and technical titles.

- *icgmsplit* (integer variable, def.=0): Setting this to 1 causes each time level of plots to be in a separate metacode file. A value of 0 causes all times to be in a single metacode file. This is true for any *ncarg_type* setting excluding "*x11*"

- *maxfld* (integer variable, def.=10): This is a parameter that is used to reserve memory for RIP. It is typically set between 10 and 15, depending on how much scratch space is needed to calculate some diagnostic fields, and how many fields are plotted in a single frame. If it is set too low, RIP will simply tell you to increase it and re-run the program.

- *itrajcalc* (integer variable, def.=0): This is a switch which turns on (1) or off (0) trajectory calculation mode. Whenever you are making plots (even trajectory

plots), it should be zero. It should only be set to 1 when you are calculating trajectories. See Chapter 6 for more details.

- *imakev5d* (integer variable, def.=0): This is a switch which turns on (1) or off (0) the Vis5D data set creation mode. Whenever you are making plots, it should be zero. It should only be set to 1 when you are producing Vis5D data. See Chapter 7 for more details.
- *ncarg_type* (not shown in above example, character variable, def.='cgm'): Output type required. Options are 'cgm', 'ps', 'pdf', 'pdfL', 'x11'. Where 'pdf' is portrait and 'pdfL' is landscape.
- *istopmiss* (not shown in above example, integer variable, def.=1): This switch determines the behavior for RIP when a user-requested field is not available. The default is to stop. Setting the switch to 0 tells RIP to ignore the missing field and to continue plotting.
- *rip_root* (not shown in above example, character variable, def.='/dev/null'): If for some reason RIP cannot access the path name that is stored in your UNIX environment variable RIP_ROOT (specifying where your RIP utility files are located), or you want to over-ride the path name specified in RIP_ROOT for a particular execution of RIP, you can specify a path name in the variable *rip_root* in the *&userin* namelist in your UIF. The default value of '/dev/null' causes RIP to try to access the path name from the environment variable RIP_ROOT, instead of trying to over-ride it.

Note: With some compilers such as Sun's Fortran 90, when assigning values to an array in a namelist, each array element must be separately referenced with its index in parentheses. This would apply to the *ptimes* and *iptimes* arrays described above. Thus, in this situation, the specification of *ptimes* in the above sample *&userin* namelist would appear as

```
ptimes(1)=0,ptimes(2)=6,ptimes(3)=12,
```

The second namelist in the UIF is called *&trajcalc*. This section is ignored by RIP if *itrajcalc*=0. Trajectory calculation mode and use of the *&trajcalc* namelist are described in Chapter 6.

b. The plot specification table

The plot specification table (PST) provides all of the user control over particular aspects of individual frames and overlays. The basic structure of the PST is as follows. The first line of the PST is a line of consecutive equal signs. This line, as well as the next two lines, is ignored–they are simply a banner that says that this is the PST. After that are several groups of one or more lines separated by a full line of equal signs. Each group of lines is a frame specification group (FSG), because it describes what will appear in a frame. A frame is defined as one frame of metacode. Each FSG must be ended with a full line of equal signs–that is how RIP knows that it has reached the end of the FSG. (Actually, RIP only looks for four consecutive equal signs, but the equal signs are continued to the end of the line for cosmetic purposes.) Each line within the FSG is a plot

specification line (PSL), because it describes what will appear in a plot. A plot is defined as one call to a major plotting routine (e.g. a contour plot, a vector plot, a map background, etc.). Hence, a FSG that has three PSLs in it will result in a frame that has three overlaid plots.

Each PSL contains several plot specification settings (PSSs), of the form

keyword = value [,value,value,...]

where *keyword* is a 4-character code word that refers to a specific aspect of the plot. Some keywords need one value, some need two, and some need an arbitrary number of values. Keywords that require more than one value should have those values separated by commas. Semicolons must separate all the PSSs within a PSL, but the final PSS in a PSL must have no semicolon after it–this is how RIP identifies the end of the PSL. Any amount of white space (i.e., blanks or tabs) is allowed anywhere in a PSS or PSL, because all white space will be removed after the line is read into RIP. The use of white space can help make your PST more readable. The order of the PSSs in a PSL does not matter, though the common convention is to first specify the *feld* keyword, then the *ptyp* keyword, and then other keywords in no particular order. A PSL may be as long as 240 characters, including spaces. However, if you want to keep all your text within the width of your computer screen, then a "greater than" symbol (>) at the end of the line can be used to indicate that the PSL will continue onto the next line. You may continue to the next line as many times as you want for a PSL, but the total length of the PSL, including spaces, cannot exceed 240 characters.

Any line in the PST can be commented out, simply by putting a pound sign (#) anywhere in the line (at the beginning makes the most sense). Note that the pound sign only comments out the line, which is not necessarily the same as the PSL. If the PSL is continued onto another line, both lines must be commented out in order to comment out the entire PSL. A partial PSL will likely cause a painful error in RIP. If all the PSLs in a FSG are commented out, then the line of equal signs at the end of the FSG should also be commented out.

There is a special keyword, *incl*, which allows the user to tell RIP to insert (at run time) additional information from another file into the plot specification table. This capability makes it easier to repeat large sections of plot specification information in a single input file, or to maintain a library of "canned" plot specifications that can be easily included in different input files. The *incl* keyword is described in more detail in Appendix A.

Each keyword has associated with it a variable in the program, and this variable may be integer, real, character, or logical. It also may be an array. The keywords that are associated with a real variable expect values that are of Fortran floating-point format. All of the following are examples of valid values:

```
1, 2, 1., 2., 1.23, 34565, -1e-13, -1.01e+16, 6.52349, -5
```

The keywords that are associated with an integer variable also expect values that are of Fortran floating point format. That is because they are initially read in as a floating-point number, and then rounded (not truncated) to the nearest integer. Hence, all of the above examples of numbers would also be valid for keywords that are associated with an integer variable (except the 8th number, which would be out of range for a 32-bit integer). The fifth and ninth values would be rounded to 1 and 7, respectively.

The keywords that are associated with a character variable expect values that are character strings. They should NOT be in single quotes, and should also not have any blank characters, commas, or semicolons in them.

The keywords that are associated with a logical variable should not have any value. They are set to .FALSE. by default, and simply the fact that the keyword appears will cause the associated variable to be set to .TRUE..

The keywords that are associated with an array (of any type) may expect more than one value. In this case, the values should be separated by commas, as mentioned above.

As an example, here is a typical PSL:

```
feld= uuu,vvv; ptyp=hv; vcor =p; levs=1000.,850,700, 500.;
vmax=15; colr=>
sky.blue; nmsg; smth= 4
```

In this example, *feld* is a keyword that has two character values, *ptyp* has one character value, *vcor* has one character value, *levs* has four real values, *vmax* has one real value, *colr* has one character value, *nmsg* is a logical flag that will cause the associated logical variable to be set to *.true.*, and *smth* has one integer value. Note the continuation character (>) at the end of the first line, indicating that the PSL continues onto the next line. There should be no semicolon at the end of the last PSS. Also in this example, a somewhat sloppy placement of blank spaces was purposefully done to demonstrate that blanks or tabs can be harmlessly placed anywhere in the PSL.

All the keywords are set to a default value prior to the reading of the plot specification table. With regard to the default setting of keywords, there are two basic types of keywords: those that "remember" their values, and those that "forget" their values. The type that remembers its value (which will subsequently be referred to as type *R*) is set to its default value only at the outset, and then it simply retains its value from one PSL to the next (and even from one FSG to the next) unless it is explicitly changed by a PSS. The type that forgets its value (which will subsequently be referred to as type *F*) is reset to its default value after every PSL. Type *R* keywords are primarily those that deal with location (e.g. the subdomain for horizontal plots, the vertical coordinate and levels for horizontal plots, cross section end points, etc.).

This chapter has described the basic rules to follow in creating the PST. Appendix A provides a description of all of the available keywords, in alphabetical order.

5. Running RIP

Each execution of RIP requires three basic things: a RIP executable, a model data set and a user input file (UIF). Assuming you have followed the procedures outlined in the previous chapters, you should have all of these. The UIF should have a name of the form *rip-execution-name.in*, where rip-execution-name is a name that uniquely defines the UIF and the set of plots it will generate. The syntax for the executable, *rip*, is as follows:

```
rip [-f] model-data-set-name rip-execution-name
```

In the above, model-data-set-name is the same model-data-set-name that was used in creating the RIP data set with the program *ripdp*. model-data-set-name may also include a path name relative to the directory you are working in, if the data files are not in your present working directory. rip-execution-name is the unique name for this RIP execution, and it also defines the name of the UIF that RIP will look for. The intended syntax is to exclude the ".in" extension in rip-execution-name. However, if you include it by mistake, RIP will recognize it and proceed without trouble. The *–f* option causes the standard output (i.e., the textual print out) from RIP to be written to a file called *rip-execution-name.out*. Without the *–f* option, the standard output is sent to the screen. The standard output from RIP is a somewhat cryptic sequence of messages that shows what is happening in the program execution.

As RIP executes, it creates either a single metacode file or a series of metacode files, depending on whether or not *icgmsplit* was set to 0 or 1 in the *&userin* namelist. If only one file was requested, the name of that metacode file is *rip-execution-name.TYPE (where TYPE could be cgm, ps, pdf)*. If separate files were requested for each plot time, they are named *rip-execution-nameA.TYPE, rip-execution-nameB.TYPE,* etc.

A common arrangement is to work in a directory that you've set up for a particular data set, with your UIFs and plot files in that directory, and a subdirectory called *data* that contains the large number of RIP data files. So, for example, if your data set was called "superstorm" and you had an input file called *sfcfields.in*, you would invoke RIP in the following manner:

```
rip data/superstorm sfcfields
```

RIP prints information to the screen as it runs, and then after it has completed execution, there will be a new file in your present working directory, *sfcfields.TYPE.* This is a metacode file containing the plots you requested. If *ncarg_type* was set to 'x11', plots will be displayed on the screen as they are being created.

You could also have the standard print information sent to the file *sfcfields.out* and run RIP in the background:

```
rip -f data/superstorm sfcfields &
```

In this case, after RIP has completed execution, you will see two new files in your present working directory. The file *sfcfields.out* contains print out from the RIP execution. The file *sfcfields.TYPE* is a metacode file containing the plots you requested.

Although the TYPE=cgm metacode file has a *.cgm* suffix, it is not a standard computer graphics metacode (CGM) file. It is an NCAR CGM file that is created by the NCAR Graphics plotting package. It can be viewed with any of the standard NCAR CGM translators, such as *ctrans*, *ictrans*, or *idt*.

6. Calculating and plotting trajectories

Because trajectories are a unique feature of RIP and require special instructions to create, this chapter is devoted to a general explanation of the trajectory calculating and plotting utility. RIP deals with trajectories in two separate steps, each of which requires a separate execution of the program.

a. Trajectory calculation

The first step is trajectory calculation, which is controlled exclusively through the namelist. No plots are generated in a trajectory calculation run. In order to run RIP in trajectory calculation mode, the variable *itrajcalc* must be set to 1 in the *&userin* namelist. All other variables in the *&userin* part of the namelist are ignored. The *&trajcalc* part of the namelist contains all the information necessary to set up the trajectory calculation run. The following is a description of the variables that need to be set in the *&trajcalc* section:

- *rtim*: the release time (in forecast hours) for the trajectories.
- *ctim*: the completion time (in forecast hours) for the trajectories.

> Note: the direction of the trajectory calculation (forward or backward) is determined by *rtim* and *ctim*. If *rtim<ctim*, trajectories are forward. If *rtim>ctim*, trajectories are backward.

- *dtfile*: the time increment (in seconds) between data files.
- *dttraj*: the time step (in seconds) for trajectory calculation. If this is set to a smaller value than *dtfile*, then velocity data are linearly interpolated in time to the trajectory time steps. Believe it or not, this actually does improve the accuracy of the trajectories. For hourly data, a ten minute (600 s) trajectory time step is often used.
- *vctraj*: the vertical coordinate of values specified for *zktraj*. *vctraj* is specified as a single character in single quotes:

> 's': *zktraj* values are model vertical level indices
> 'p': *zktraj* values are pressure values, in mb
> 'z': *zktraj* values are height values, in km
> 'm': *zktraj* values are temperature values, in C

‘t’: *zktraj* values are potential temperature values, in K

‘e’: *zktraj* values are equivalent potential temperature values, in K

- *ihydrometeor*: a flag which, if set to 1, causes the trajectory calculation algorithm to use the hydrometeor fall speed (an average fall speed weighted by the mixing ratios of the different precipitation types) instead of the vertical air velocity. The purpose of this flag is to produce hydrometeor trajectories instead of air parcel trajectories.
- *xjtraj,yitraj*: real arrays containing *x* and *y* values (in grid points) of the initial positions of the trajectories. The grid values can be non-integer values, and should be relative to the grid position at the trajectory release time (this is important for trajectories in moving nests).
- *zktraj*: real array containing values of the vertical location of the initial points of the trajectories. The type of value used (e.g. pressure, height, etc.) must be consistent with *vctraj*. If *vctraj* is ‘*s*’, then the values can be specified either as σ values (numbers between 0 and 1) or as *k* indices (integers greater than or equal to 1).

It is also possible to define a 3D array of trajectory initial points, without having to specify the [x,y,z] locations of every point. The grid can be of arbitrary horizontal orientation. To define the grid, you must specify the first seven values of xjtraj as follows: The first two values should be the x and y values of the lower left corner of the trajectory horizontal grid. The next two values should be the x and y values of another point defining the positive x-axis of the traj. grid (i.e., the positive x-axis will point from the corner point to this point). The fifth value should be the trajectory grid spacing, in model grid lengths. The final two values should be the number of points in the x and y directions of the trajectory horizontal grid. The first value of xjtraj should be negative, indicating that a grid is to be defined (rather than just individual points), but the absolute value of that value will be used. Any yitraj values given are ignored. The zktraj values specify the vertical levels of the 3D grid to be defined. Note that any vertical coordinate may still be used if defining a 3D grid of trajectories.

If no diagnostic quantities along the trajectories are desired, the plot specification table is left blank (except that the first three lines comprising the "Plot Specification Table" banner are retained). If diagnostic quantities are desired, they can be requested in the plot specification table (although no plots will be produced by these specifications, since you are running RIP in trajectory calculation mode). Since no plots are produced, only a minimum of information is necessary in the plot specification table. In most cases, only the *feld* keyword needs to be set. For some fields, other keywords that affect the calculation of the field should be set (such as *strm*, *rfst*, *crag*, *crbg*, *shrd*, *grad*, *gdir*, *qgsm*, *smcp*, and *addf*). Keywords that only affect how and where the field is plotted can be omitted. Any of the diagnostic quantities listed in Appendix B can be calculated along trajectories, with the exception of the Sawyer-Eliassen diagnostics. Each desired diagnostic quantity should be specified in its own frame specification group (FSG) (i.e. only one *feld*= setting between each line of repeated equal signs). The only exception to this is if you are using the *addf* keyword. In that case, all of the plot specification lines

(PSLs) corresponding to the fields being added (or subtracted) should be in one FSG. As a simple example, if you want to calculate cloud water mixing ratio, pressure, omega, and relative humidity along your trajectories, your plot specification table would look like this:

```
===============================================================
--------------- Plot Specification Table -----------------
===============================================================
feld=qcl
===============================================================
feld=prs
===============================================================
feld=omg
===============================================================
feld=rhu
===============================================================
```

Once the input file is set up, RIP is run as outlined in Chapter 5. Since no plots are generated when RIP is run in trajectory calculation mode, no *rip-execution-name.TYPE* file is created. (*rip-execution-name* is the unique name you've chosen for a particular execution of RIP. See Chapter 5). However, two new files are created that are not in a regular (non-trajectory-calculation) execution of RIP. The first is a file that contains the positions of all the requested trajectories at all the trajectory time steps, called *rip-execution-name.traj*. The second is a file that contains requested diagnostic quantities along the trajectories at all data times during the trajectory period, called *rip-execution-name.diag*. The *.diag* file is only created if diagnostic fields were requested in the plot specification table.

b. Trajectory plotting

Once the trajectories have been calculated, they can be plotted in subsequent RIP executions. Because the plotting of trajectories is performed with a different execution of RIP than the trajectory calculation, the plotting run should have a different rip-execution-name than any previous trajectory calculation runs.

Trajectories are plotted by including an appropriate PSL in the PST. There are three keywords that are necessary to plot trajectories, and several optional keywords. The necessary keywords are *feld*, *ptyp*, and *tjfl*. *feld* should be set to one of five possibilities: *arrow*, *ribbon*, *swarm*, *gridswarm*, or *circle* (these fields are described in detail below). *ptyp* should be set to either *ht* (for "horizontal trajectory plot") or *vt* (for "vertical (cross section) trajectory plot"). *tjfl* tells RIP which trajectory position file you want to access for the trajectory plot.

As mentioned above, there are four different representations of trajectories, as specified by the *feld* keyword:

- *feld=arrow*: This representation shows trajectories as curved arrows, with arrowheads drawn along each trajectory at a specified time interval. If the plot is a horizontal trajectory plot (*ptyp=ht*), the width of each arrowhead is proportional to the height of the trajectory at that time. If the plot is a vertical (cross section) trajectory plot (*ptyp=vt*), the width of each arrowhead is constant. The arrowhead that corresponds to the time of the plot is boldened.
- *feld=ribbon*: This representation shows trajectories as curved ribbons, with arrowheads drawn along each trajectory at a specified time interval. If the plot is a horizontal trajectory plot (*ptyp=ht*), the width of each arrowhead, and the width of the ribbon, is proportional to the height of the trajectory at that time. If the plot is a vertical (cross section) trajectory plot (*ptyp=vt*), the width of each arrowhead (and the ribbon) is constant. The arrowhead that corresponds to the time of the plot is boldened.
- *feld=swarm*: This representation shows a group of trajectories attached to each other by straight lines at specified times. The trajectories are connected to each other in the same order at each time they are plotted, so that the time evolution of a material curve can be depicted. Swarms can be plotted either as horizontal or vertical trajectory plots (*ptyp=ht* or *ptyp=vt*).
- *feld=gridswarm*: This is the same as *swarm*, except it works on the assumption that part or all of the trajectories in the position file were initially arranged in a row-oriented 2-D array, or "gridswarm". The evolution of this gridswarm array is depicted as a rectangular grid at the initial time, and as a deformed grid at other specified times. The gridswarm being plotted can have any orientation in 3D space, although the means to create arbitrarily oriented gridswarms when RIP is used in trajectory calculation mode are limited. Creative use of the "3D grid of trajectories" capability described above under the description of *zktraj* can be used to initialize horizontal gridswarms of arbitrary horizontal orientation (but on constant vertical levels).
- *feld=circle*: This representation shows the trajectories as circles located at the positions of the trajectories at the current plotting time, in which the diameter of the circles is proportional to the net ascent of the trajectories (in terms of the chosen vertical coordinate) during the specified time interval. It is only available as a horizontal trajectory plot (*ptyp=ht*).

The optional keywords that affect trajectory plots are listed below. The functionality of some of these keywords may depend on what type of trajectory representation (as specified by the *feld* keyword) you are using. See Appendix A, "Keywords", for more details.

colr: Color of trajectories, swarms, or circles representing positive ascent.

cong: Color of circles representing negative ascent.

dash: Dash pattern of trajectories, swarms, or circles representing positive ascent.

dang: Dash pattern of circles representing negative ascent.

lchl: Label color for storm position (for storm-relative trajectories).

lcll: Label color for trajectory labels.

linw: Line width of trajectories, swarms, or circles representing positive ascent.

lwng: Line width of circles representing negative ascent.

nmsg: No trajectory height legend.

nohl: No storm position marker (for storm-relative trajectories).

nolb: No trajectory labels.

strm: Storm velocity (for storm-relative trajectories).

tjar: Arrow widths for trajectories or circles.

tjen: End time (in forecast hour) of plotted trajectories or swarms, or end time for net ascent calculation (for circles).

tjfl: File name of trajectory position info.

tjid: ID numbers of trajectories to be plotted or swarm points to be connected, or gridswarm definition.

tjsp: Storm position for storm-relative trajectories.

tjst: Start time (in forecast hour) of plotted trajectories or swarms, or start time for net ascent calculation (for circles).

tjti: Time interval for trajectory arrow heads, or for swarms, in hours.

tshl: Text size for storm position marker (for storm-relative trajectories).

tslb: Text size for trajectory labels

vcor: Vertical coordinate for trajectory height legend or for net ascent calculation (for circles). (for *ptyp=ht*).

vwin: Vertical window for trajectory height legend, or reference vertical displacement for circles.

A simple example of an FSG that includes a trajectory plot follows:

```
===========================================================
--------------- Plot Specification Table ----------------
===========================================================
feld=the; ptyp=hc; vcor=p; levs=850; cint=2
feld=arrow; ptyp=ht; tjfl=tjcalc3.traj; tjid=3,-12,3; >
    vcor=z; colr=green; tjst=12; tjen=18
feld=map; ptyp=hb
feld=tic; ptyp=hb
===========================================================
```

In the above, the first line specifies that potential temperature contours at 850 mb will be plotted. The second line specifies that "arrow-type" trajectories will be plotted. The trajectories will come from the trajectory position file called *tjcalc3.traj*, created by a previous execution of RIP in trajectory calculation mode. The trajectories will be plotted in green, and only the parts from forecast hours 12 through 18 will be plotted (i.e. you can plot a subset of the time period for which the trajectories were calculated). Only trajectory numbers 3, 6, 9, and 12 will be plotted, as specified by the *tjid* keyword. Finally, a map and tick mark background will be plotted.

c. Printing out trajectory positions

Sometimes, you may want to examine the contents of a trajectory position file. Since it is a binary file, the trajectory position file cannot simply be printed out. However, a short program is provided in the *src/* directory in the RIP tar file called *showtraj.f,* which reads the trajectory position file and prints out its contents in a readable form. The program should have been compiled when you originally ran *make*, and when you run *showtraj*, it prompts you for the name of the trajectory position file to be printed out.

d. Printing out diagnostics along trajectories

As mentioned above, if fields are specified in the plot specification table for a trajectory calculation run, then RIP produces a *.diag* file that contains values of those fields along the trajectories. This file is an unformatted Fortran file, so another program is required to view the diagnostics. Among the Fortran files included in the *src/* directory in the RIP tar file is *tabdiag.f*, which serves this purpose. It is also compiled when *make* is run.

In order to use the program, you must first set up a special input file that contains two lines. The first line should be the column headings you want to see in the table that will be produced by *tabdiag*, with the entire line enclosed in single quotes. The second line is a Fortran I/O format string, also enclosed in single quotes, which determines how the diagnostic values are printed out. An example of an input file for *tabdiag* is included in the RIP tar file, called *tabdiag.in*, and is shown below.

```
' Time (h) qcl (g/kg) Press. (mb) Omega (ubar/s) RH (%) '
'(5(3x,f9.3,3x))'
```

If a trajectory calculation run requested that *qcl, prs, omg,* and *rhu* be calculated along the trajectories, this input file for *tabdiag* might be used to tabulate that data. Note that the *tabdiag* input file should include a column heading and Fortran formatting for the time, in hours, in the first column. The program *tabdiag* always prints out the time in the first column, regardless of which diagnostic fields were requested. Once the input file is set up, *tabdiag* is run as follows:

```
tabdiag diagnostic-output-file tabdiag-input-file
```

The result will be a text file with a table for each trajectory, showing the time evolution of the diagnostic quantities. Some adjustment of the column headings and format statement will probably be necessary to make it look just right.

7. Creating a data set for Vis5D

Vis5D is a powerful visualization software package developed at the University of Wisconsin, and is widely used by mesoscale modelers to perform interactive 3D visualization of model output. Although it does not have the flexibility of RIP for producing a wide range of 2D plot types with extensive user control over plot details, its 3D visualization capability and fast interactive response make it an attractive complement to RIP.

A key difference between RIP and Vis5D is that RIP was originally developed specifically for scientific diagnosis and operational display of mesoscale modeling system output. This has two important implications: (1) The RIP system can ingest model output files, and (2) RIP can produce a wide array of diagnostic quantities that mesoscale modelers want to see. Thus, it makes sense to make use of these qualities to have RIP act as a bridge between a mesoscale model and the Vis5D package. For this reason, a Vis5D-format data-generating capability was added to RIP. With this capability, you can create a Vis5D data set from your model data set, including any diagnostic quantities that RIP currently calculates.

The Vis5D mode in RIP is switched on by setting *imakev5d=1* in the *&userin* namelist in the UIF. All other variables in the *&userin* part of the namelist are ignored. No plots are generated in Vis5D mode. The desired diagnostic quantities are specified in the plot specification table (PST). Since no plots are produced, only a minimum of information is necessary in the plot specification table. In most cases, only the *feld* keyword needs to be set, and vertical levels should be specified with *levs* (in km) for the first field requested. The vertical coordinate will automatically be set to *'z'*, so there is no need to set *vcor=z*. The levels specified with *levs* for the first requested field will apply to all 3D fields requested, so the *levs* specification need not be repeated for every field. You are free to choose whatever levels you wish, bearing in mind that the data will be interpolated from the data set's vertical levels to the chosen height levels.

For some fields, other keywords that affect the calculation of the field should be set (such as *strm*, *rfst*, *crag*, *crbg*, *shrd*, *grad*, *gdir*, *qgsm*, *smcp*, and *addf*). Keywords that only

affect how and where the field is plotted can be omitted. Any of the diagnostic quantities listed in Appendix B can be added to the Vis5D data set, with the exception of the Sawyer-Eliassen diagnostics. Each desired diagnostic quantity should be specified in its own frame specification group (FSG) (i.e. only one *feld=* setting between each line of repeated equal signs). The only exception to this is if you are using the *addf* keyword. In that case, all of the plot specification lines (PSLs) corresponding to the fields being added (or subtracted) should be in one FSG.

As a simple example, if you want to create a Vis5D data set with velocities, pressure, sea-level pressure, potential temperature, cloud mixing ratio, precipitation mixing ratio, potential vorticity, and water vapor mixing ratio, your plot specification table would look something like this:

```
===============================================================
--------------- Plot Specification Table -----------------
===============================================================
feld=uuu; levs=0,-12,.5
===============================================================
feld=vvv
===============================================================
feld=www
===============================================================
feld=prs
===============================================================
feld=slp
===============================================================
feld=the
===============================================================
feld=qcl
===============================================================
feld=qpr
===============================================================
feld=pvo
===============================================================
feld=qvp
===============================================================
```

Once the user input file is set up, RIP is run as outlined in Chapter 5. Since no plots are generated when RIP is run in Vis5D mode, no *rip-execution-name.TYPE* file is created. (*rip-execution-name* is the unique name you've chosen for a particular execution of RIP. See Chapter 5). However, a file is created with the name *rip-execution-name.v5d*. This file is the Vis5D data set, which can be used by the Vis5D program to interactively display your model data set.

The map projection information will automatically be generated by RIP and included in the Vis5D data set. Therefore, you don't have to explicitly request *feld=map* in the plot specification table. However, there are some complications with converting the map

background, as specified in RIP, to the map background parameters required by Vis5D. Currently, RIP can only make the conversion for Lambert conformal maps, and even that conversion does not produce an exact duplication of the correct map background.

Vis5D also has its own terrain data base for producing a colored terrain-relief map background--you don't need to specifically request *feld=ter* to get this. However, if you want to look at the actual model terrain as a contour or color-filled field, you should add *feld=ter* to your plot specification table.

8. Other special situations

a. 2-D model output

Some models (such as WRF) can be run in a 2-D mode. In such a case, the domain dimension in the *y*-direction will be 3 "dot" (*u/v)* grid points and 2 "cross" (*T/p*) grid points. The only type of plots that make sense in this situation are vertical cross sections (*ptyp=vc, vv, vw, vt,* or *vb*). The *y*-value of the cross section endpoints (as specified by the second value given for *crsa* and *crsb*) should always be "2".

Appendix A. Keywords

This appendix is a list of all the available keywords. Each entry contains the keyword, a brief statement of what the keyword is, the type and number of values expected, the default values, whether they are type R ("remembered") or F ("forgotten"), the purpose of the keyword, and, in some cases, a more extended explanation of how to use the keyword.

The most important keywords, which must be specified for every plot (and are typically specified first), are *feld* and *ptyp*. These are described first. Following these two is an alphabetic list of all other keywords.

feld: Field to be plotted

> Expects 1, 2, or 3 character values, each up to 10 characters in length; defaults are *dm1, dm2,* and *dm3* (these are nonsense values); type is *F*.

> Purpose: This specifies the field or fields for the plot. For contour plots, character plots, skew-T temperature trace plots, trajectory plots, vertical profiles, and background plots, only one value is expected. For vector plots that require two components (if *ptyp* is set to *hv, hs, vw,* or *sv*), two fields are expected. For vector plots that require three components (if *ptyp* is set to *vv*), three fields are expected. The available fields are described in Appendix B.

> In addition to the available fields for plotting described in Appendix B, *feld* can also be set to the ending part of any ripdp data file name to plot the contents of

that data file. This capability is useful for plotting variables that RIPDP encountered (but did not expect) in the model output. It also allows for the plotting of diagnostic quantities that have been saved to a file with a user-specified variable name, using the *save* keyword. For example, if there is a RIPDP data file called *mycase_012.00000_CATSANDDOGS*, it can be plotted by using *feld=CATSANDDOGS*.

ptyp: Plot type

Expects 1 character value of length 2; default is 'hc'; type is *F*.

Purpose: This keyword determines what type of plot will be drawn for this PSL. There are currently 11 possibilities:

hc: horizontal contour plot
hv: horizontal vector plot
hs: horizontal streamline plot
hh: horizontal character plot (for things like land use, snow cover)
ht: horizontal trajectory plot
hb: horizontal background plot (maps, perimeter tick marks)
vc: vertical (cross section) contour plot
vv: vertical vector plot (circulation vectors in the plain of the cross sec.)
vw: vertical wind vector plot (hor. wind vectors in a vert. cross sec.)
vt: vertical trajectory plot (trajectories projected onto the plane of the cross section)
vb: vertical background plot (perimeter tick marks and ground curve for cross section)
sc: sounding contour plot (a temperature trace on a skew-T)
sv: sounding vector plot (a wind barb column on a skew-T)
sb: sounding background plot (the background skew-T grid)
pc: vertical profile contour plot (a trace of the specified variable as a function of height)

A few notes on vertical (cross section) plots: The endpoints of the cross section are specified with keywords *crsa* and *crsb* (see description of these keywords below). The data are interpolated in the horizontal direction from the grid to an arbitrary number of equally spaced points along the cross section. That number is automatically chosen by RIP for contour plots, but you can specify it for vector plots. In the vertical direction, the data are not interpolated. Rather, the plotting grid itself is transformed so that the model vertical levels are correctly placed with respect to whatever vertical coordinate has been chosen for the cross section. The tick mark background plot will also draw a curve indicating the position of the ground.

addf: Add (or subtract) this field to (or from) the next field

Expects 1 real value; default is 0.0, which means do not add this field to the next field; type is *F*.

Purpose: This option allows you to add fields together prior to plotting. Added fields should appear as separate PSLs. In PSLs that have the *addf* keyword,

usually only the *feld* keyword needs to be set. For some fields, other keywords that affect the calculation of the field should be set (such as *strm*, *rfst*, *crsa*, *crsb*, *shrd*, *grad*, *gdir*, *qgsm*, *redo*, *sepa*, *lapl*, *hadv*, *diff*, and *smcp*). PSLs that only affect how and where the field is plotted can be omitted. In the final field to be added, *addf* should be omitted, and all the other desired keywords should be set.

As an example, suppose you want to plot the sum of cloud water, cloud ice, rain, and snow. This would be accomplished as follows:

```
feld=qcw; ptyp=hc; addf=1
feld=qci; ptyp=hc; addf=1
feld=qra; ptyp=hc; addf=1
feld=qsn; ptyp=hc; vcor=p; levs=850; cint=1
```

Each field is multiplied by its value of *addf* prior to adding. Hence, one can add or subtract a field or a multiple of a field. For example, you could plot ageostrophic wind vectors as follows (although this field exists, called *uageo* and *vageo*):

```
feld=ugeo,vgeo; ptyp=hv; addf=-1
feld=uuu,vvv; ptyp=hv; vcor=p; levs=500; intv=3
```

arng: Automatic range

Expects no values (logical); default is .false.; type is *F*.

Purpose: This flag turns on the automatic range feature. The automatic range feature changes the way in which the *cosq* keyword works for color filling. If automatic range is on, then the values in the color sequence should be made to range from 0 to 100, regardless of the units or the actual range of the data in the plotted field. RIP then determines the minimum and maximum values in the plotted field, and linearly maps the minimum-to-maximum range on the 0-to-100 range. Hence, the full color range that is defined will automatically cover the exact range of data in the plotted field. Note that if you use *arng*, specification of contour interval (*cint*), beginning contour (*cbeg*), and ending contour (*cend*) should still be specified in terms of the actual values of the field if you need to use these keywords.

Example:
```
feld=tmc; ptyp=hc; cmth=fill; arng;
cosq=0,blue,50,white,100,red
```

axld: Large labeled tick increment along distance axis

Expects 1 real value; default is 100.; type is *F*.

Purpose: This specifies how often (in units of km) there should be large labeled tick marks along the distance axis of a cross section if *feld=tic* is specified. A value of 0 indicates no large labeled tick marks should be drawn.

axlg: Large labeled tick increment for horizontal plots

Expects 1 integer value; default is 10; type is *F*.

Purpose: This specifies how often (in grid points) there should be large labeled tick marks in a horizontal plot if *feld=tic* is specified. A value of 0 indicates no large labeled tick marks should be drawn.

axlv: Large labeled tick increment along vertical axis

Expects 1 real value; default depends on vertical coordinate; type is *F*.

Purpose: This specifies how often (in units of the vertical coordinate) there should be large labeled tick marks along the vertical axis of a cross section if *feld=tic* is specified. A value of 0 indicates no large labeled tick marks should be drawn. The default values are 10 for *vcor=s* (model vertical level index), 100 mb for *vcor=p, l,* or *x* (pressure, log pressure, or Exner function), 1.0 km for *vcor=z* (height) , 10 K for *vcor=t* or *e* (potential temperature or equivalent potential temperature), 10 C for *vcor=m* (temperature), and 1.0 PVU for *vcor=q* (potential vorticity).

axtd: Small tick increment along distance axis

Expects 1 real value; default is 10.; type is *F*.

Purpose: This specifies how often (in units of km) there should be small tick marks along the distance axis of a cross section if *feld=tic* is specified. A value of 0 indicates no small tick marks should be drawn.

axtg: Small tick increment for horizontal plots

Expects 1 integer value; default is 1; type is *F*.

Purpose: This specifies how often (in grid points) there should be small tick marks in a horizontal plot if *feld=tic* is specified. A value of 0 indicates no small tick marks should be drawn.

axtv: Small tick increment along vertical axis

Expects 1 real value; default depends on vertical coordinate; type is *F*.

Purpose: This specifies how often (in units of the vertical coordinate) there should be small labeled tick marks along the vertical axis of a cross section if *feld=tic* is specified. A value of 0 indicates no small labeled tick marks should be drawn. The default values are 1 for *vcor=s* (model vertical level index), 10 mb for *vcor=p, l,* or *x* (pressure, log pressure, or Exner function), 0.1 km for *vcor=z* (height), 1 K for *vcor=t* or *e* (potential temperature or equivalent potential temperature), 1 C for *vcor=m* (temperature), and 0.1 PVU for *vcor=q* (potential vorticity).

bogs: Flag to output a bogus sounding

Expects no values (logical); default is .false.; type is *F*.

Purpose: If this is true, RIP will output a bogus sounding print out in "little-r" format to the file *fort.66*.

cbeg: Beginning contour value

Expects 1 real value; default is -9.0e9 (nine times ten to the ninth); type is *F*.

Purpose: This specifies the beginning contour value. A value of -9.0e9 causes RIP to choose a nice value that is near the minimum value in the plotted field. A value of 9.0e9 causes RIP to choose a nice value that is near the maximum value in the plotted field. RIP generates contours by starting at *cbeg* and stepping toward *cend* by the amount *cint*, until the value of *cend* is reached. This stepping process may go in either direction, i.e. *cint* may be greater or less than *cbeg*. *cbeg* is also used to specify the lower limit of the horizontal axis for vertical profiles (*ptyp=pr*).

cend: Ending contour value

Expects 1 real value; default is 9.0e9 (nine times ten to the ninth); type is *F*.

Purpose: This specifies the ending contour value. A value of -9.0e9 causes RIP to choose a nice value that is near the minimum value in the plotted field. A value of 9.0e9 causes RIP to choose a nice value that is near the maximum value in the plotted field. *cend* is also used to specify the upper limit of the horizontal axis for vertical profiles (*ptyp=pr*).

chfl: Character fill flag

Expects no values (logical); default is .false.; type is *F*.

Purpose: If this is true, then the character plotting routine will not plot characters, but instead will fill each grid box with the appropriate color from the color sequence, using cell filling.

cint: Contour interval (contour plots) or lat/lon line interval (for maps)

Expects 1 real value; default for contour plots is a nice value that generates near the desired number of contours; default for maps is 10; type is *F*

Purpose: For contour plots, this specifies the contour interval. If it is not specified, RIP will choose a nice value that generates near the desired number of contours (as specified by the *ncon* keyword). For maps, *cint* specifies the contour interval for latitude and longitude lines on a map background.

cmth: Contouring method

> Expects 1 character value; default is 'cont'; type is *F*.

> Purpose: Determines whether this plot will generate contour lines (*cont*), contour filling with "area fill" (*fill*), both contours and contour filling with "area fill" (*both*), contour filling with "cell fill" (*cell*), or both contours and contour filling with "cell fill" (*ceco*).

> Note: There are two ways of making color-filled contour plots. One is with "area fill", in which the regions between contours are defined as polygonal areas which are filled with appropriate colors. The other is with "cell fill", in which each grid point is assigned a square "cell" defined as as the surrounding grid box, and the cell is assigned a color based on the field value at the grid point. Cell-filling is a new addition to RIP. Its advantage is that plots are generated considerably faster than with area filling, especially for large domains. It's disadvantages are that it can look a bit "pixely" for coarser domains; it is not compatible with the "transparent color" feature (see keyword *cosq*); and it produces strange behavior in the viewing program *idt,* namely, very slow refreshing of the plot when the zoom feature is used.

coll: Labeled contour color

> Expects 1 character value; default is the value of *colr*; type is *F*.

> Purpose: This sets the color of the labeled contours, if it is to be different from *colr*. Any color name that is defined in the color table is valid.

colr: Plot color

> Expects 1 character value; default is 'def.foreground'; type is *F*.

> Purpose: This is the main color that will be used for the plot. Any color name that is defined in the color table is valid. Depending on what type of plot is being drawn, *colr* determines the color of contours, vectors, wind barbs, streamlines, characters (in character plots), trajectories, swarms, trajectory net ascent circles, lat/lon lines (in map plots), the perimeter square and tick marks, lines (for *feld=line* or *feld=box*), bullets, or station IDs. It also sets the color of the corresponding plot title at the top of the plot, as well as the corresponding messages at the bottom of the plot.

cong: Negative contour color

> Expects 1 character value; default is the value of *colr*; type is *F*.

> Purpose: This sets the color of the negative contours, if it is to be different from *colr*. Also, for horizontal trajectory plots, if *feld=circle* is set, this color is used for circles representing negative net height changes. Any color name that is defined in the color table is valid.

conl: Negative labeled contour color

> Expects 1 character value; default is the value of *cong*; type is *F*.

> Purpose: This sets the color of the negative labeled contours, if it is to be different from *cong*. Any color name that is defined in the color table is valid.

cord: Contour ordenal specification for color sequence

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: This flag turns on the contour ordenal specification for the color sequence. This feature changes the way in which the *cosq* keyword works for color filling. If *cord* is on, then the values in the color sequence should be made to range from 1 to the number of contour color bands expected (which is always one larger than the number of contours). RIP then chooses colors for the contour color bands by interpolating in terms of contour band ordenal, rather than contour band value. Note that if you use *cord*, specification of contour interval (*cint*), beginning contour (*cbeg*), and ending contour (*cend*) should still be specified in terms of the actual values of the field if you need to use these keywords.

cosq: Color sequence

> Expects an arbitrary number of pairs of real and character values; Default is such that all real values correspond to the color *def.foreground*; type is *F*.

> Purpose: For contour plots, the color sequence determines how colors are chosen for color filled contours. For horizontal character plots, the color sequence is used for coloring the characters in character plots. The color sequence is comprised of an arbitrary number of pairs, each of which contains a real value and a color name. The values correspond to values of the plotted field The color names can be any of the names that are available in the color table.

> Examples:

> For color-filled contour plots:

> cosq=-5,white,3.5,red,7,orange,12,yellow,15,green,20,blue,24.5,violet

> The way in which colors for particular areas or cells are assigned is as follows. Each of the areas that are bounded by two different contours, or each cell whose grid value is bounded by two contour values, is assigned a discrete value that is equal to the average of the values of the bounding contours. For the areas or cells that are above the highest contour level (or below the lowest contour level), the discrete value is chosen to be half a contour interval above (or below) the bounding contour.

> The discrete value is then compared to the color sequence in order to find the two color sequence pairs whose values bracket the discrete value. Interpolated values of red, green, and blue fractions are calculated based on the discrete value, the

bracketing color sequence values, and the bracketing color sequence colors. This interpolated color is then used to fill the area or cells.

If you want some contour-bounded areas of the plot to appear transparent, so that other fields can be seen underneath, the color *transparent* can be specified in the color sequence. For a given contour-bounded area, if either of the bracketing values in the color sequence is assigned the color *transparent*, that contour-bounded area will be transparent. Note that *transparent* only works for area filling, not for cell filling.

For character plots:

The color name refers to the color that you want a particular value to be plotted in, and it can be any of the names that are available in the color table. An example of a *cosq* setting for a character plot might be

feld=xlus; ptyp=hh; cosq=1,red,2,blue,4,pink,5,yellow,>

7,sky.blue,8,violet,6,white,10,dark.blue

This would result in a plot in which, at every cross point, the land use category number would be plotted in the color as specified above. Any numbers that do not have a color specified would be plotted in the default foreground color (such as 3, 9, 11, etc., in the above example). Also, if keyword *chfl* is set, grid boxes are filled with the chosen color (using cell filling), rather than values printed in each box.

cozr: Zero contour color

Expects 1 character value; default is the value of *colr*; type is *F*.

Purpose: This sets the color of the zero contour, if it is to be different from *colr*. Any color name that is defined in the color table is valid.

crsa: Left cross section end point location

Expects a variety of values; defaults are 3,3; type is *R*.

Purpose: For vertical cross section plots, this sets the location (in the dot point domain) of the first (left) end point of a cross section. It also applies to some horizontal background plots: it sets the lower left corner of a box (*feld=box*), the first end point of a line (*feld=line*), or the location of a bullet (*feld=bull*). Finally, it is also used to define a vector from *crsa* to *crsb*, which may be used to define the direction of a vector component for keywords *strm*, *shrd*, and *gdir*.

There are four different ways to specify location. The first is in terms of the *x,y* location on the model grid. To specify location in this way, use two real values, separated by a comma (i.e. x,y). The second way is in terms of latitude and longitude. To specify location in this way, use two real values, each with the words "lat" or "lon" immediately after the number, and separated by a comma. The third way is in terms of the 5-digit WMO number for a station in North America. To specify the location in this way, supply the WMO number as a single integer value. The fourth way is in terms of a four-character ICAO station identifier for a station that has such an identifier listed in the *stationlist*. To

specify the location in this way, supply the identifier as one string, and use only upper case for the string. Examples follow.

x,y format: crsa=20.4,30;

lat,lon format: crsa=40.24lat,-78.1lon;

WMO format: crsa=72240;

character ID format: crsa=ORD;

An important limitation to be aware of is that, regardless of the format that is used, the resulting locations must be greater than 1.5, and less than mjx-.5 for x points and miy-.5 for y points.

crsb: Right cross section end point location

Expects a variety of values; defaults are 3,3; type is *R*.

Purpose: For vertical cross section plots, this sets the location (in the dot point domain) of the second (right) end point of a cross section. It also applies to some horizontal background plots: it sets the upper-right corner of a box (*feld=box*) or the second end point of a line (*feld=line*). Finally, it is also used to define a vector from *crsa* to *crsb*, which may be used to define the direction of a vector component for keywords *strm*, *shrd*, and *gdir*.

cval: Interval values

Expects a string of interval values; no default; type is *R*.

Purpose: Use to manually specify the interval levels that will be used in contour plots.

dall: Labeled contour dash pattern

Expects 1 integer value; default is the value of *dash*; type is *F*.

Purpose: This sets the dash pattern of the labeled contours, if it is to be different from *dash*.

dang: Negative contour dash pattern

Expects 1 integer value; default is the value of *dash*; type is *F*.

Purpose: This sets the dash pattern of the negative contours, if it is to be different from *dash*. Also, for horizontal trajectory plots, if *feld=circle* is set, this dash pattern is used for circles representing negative net height changes.
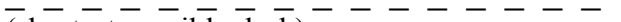
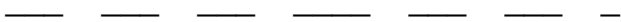*danl*: Negative labeled contour dash pattern

Expects 1 integer value; default is the value of *dang*; type is *F*.

Purpose: This sets the dash pattern of the negative labeled contours, if it is to be different from *dang*.

dash: Dash pattern

Expects one two-digit integer value; default is 10, except if *feld=map*, default is 32; type is *F*.

Purpose: Depending on the type of plot being drawn, this sets the dash pattern for contour lines, lat/lon lines (for maps), boxes and lines (*ptyp=box* or *line*), vertical bars in cross sections (*feld=vbar*), a temperature trace in a skew-T sounding, trajectories, swarms, or trajectory net ascent circles. The specification is based on the idea that a segment of a contour can be broken up into 16 pieces. The first digit of dash specifies how many solid pieces should be drawn, and the second digit specifies how many blank pieces should be drawn. If the two digits do not add up to 16, then the pattern is repeated until a pattern of 16 pieces is achieved. If the two digits add up to more than 16, then the pattern is cut off at 16. The 16-piece pattern is then used as the repeating pattern for contour lines. Again, this is best seen by example:

Dash value	This is what 2 repetitions of the 16-piece pattern look like:
10	――――――――――――――――――――― (solid)
20	――――――――――――――――――――― (same as previous pattern)
80	――――――――――――――――――――― (same as previous pattern)
11	– – – – – – – – – – – – – – – – (shortest possible dash)
21	— — — — — — — — — — — –
32	―― ―― ―― ―― ―― ―― –
88	―――――― ――――――
17	– – – –

You should experiment to determine the length of dash pattern that is obtained for various settings of *dash*.

dazr: Zero contour dash pattern

Expects 1 integer value; default is the value of *dash*; type is *F*.

Purpose: This sets the dash pattern of the zero contour, if it is to be different from *dash*.

diff: Difference fields

Expects one to three values, which are either real or character, separated by a comma(s); the order is not important; Default is no differencing; type is *F*.

Purpose: You can subtract from the specified field the same field at a different time and/or in a different data set. If you want to subtract the same field at a different time in the same data set, set *diff* equal to a real value, which is a time in hours from the beginning of the forecast or analysis. The time specified will be interpreted as an absolute time. However, if you want to specify a time relative to the current plotted time (either positive or negative is acceptable), also include the character string "rel" (no quotes) as an additional value for the *diff* keyword. If you want to subtract the same field at the same time in a different model data set, set *diff* equal to a character value, which is the prefix for the data set you want to subtract. This should be specified as a path name relative to your current working directory, the same as is model-data-set-name on the RIP command line. If you want to subtract the same field at a different time and in a different data set, set *diff* equal to a real and a character value separated by a comma, where the real value is the time in hours and the character value is the name of the data set to subtract. These can be specified in either order: time,data-set or data-set,time.

Examples:

```
diff=10;              This would subtract the same field in the
same data set
                      but at hour 10.
diff=-3,rel;          This would subtract the same field in the
same data set
                      but 3 h prior to the current time.
diff=data/noflux;     This would subtract the same field at the
same time
                      but from the data set "noflux" in the
"data" directory.
diff=rel,-6,data/dryrun;  This would subtract the same field
                          but from 6 h prior to the current time,
and from the
                          data set "dryrun" in the "data"
directory.
```

fchl: Fill color for high/low label boxes in contour plots

Expects 1 character value; default is the value of *fclb*; type is *F*.

Purpose: This sets the fill color for high/low label boxes, if it is to be different from *fclb*. Any color name that is defined in the color table is valid.

fclb: Fill color for label boxes in contour plots

Expects 1 character value; default is '999999', which means "do not fill"; type is *F*.

Purpose: This sets the fill color for label boxes. Any color name that is defined in the color table is valid.

fclo: Fill color for low label boxes in contour plots

Expects 1 character value; default is the value of *fchl*; type is *F*.

Purpose: This sets the fill color for low label boxes, if it is to be different from *fchl*. Any color name that is defined in the color table is valid.

fcnl: Fill color for negative contour label boxes in contour plots

Expects 1 character value; default is the value of *fclb*; type is *F*.

Purpose: This sets the fill color for negative contour label boxes, if it is to be different from *fclb*. Any color name that is defined in the color table is valid.

fczr: Fill color for zero contour label boxes in contour plots

Expects 1 character value; default is the value of *fclb*; type is *F*.

Purpose: This sets the fill color for zero contour label boxes, if it is to be different from *fclb*. Any color name that is defined in the color table is valid.

fulb: Full barb value

Expects 1 character value; default is '5mps'; type is *F*.

Purpose: Sets the value of a full wind barb on either horizontal wind barb plots, vertical cross sections with horizontal wind barbs, or wind barb columns in skew-T sounding plots. If you want to follow the convention used on standard 20th-century (non-SI units) U.S. weather charts, the appropriate value would be '10kts' (which refers to 10 knots). The other choices are '10mps' or '5mps' (for "10 meters per second" of "5 meters per second", respectively). A message will appear on the plot indicating the choice of *fulb*.

gdir: Direction for horizontal gradient, Laplacian, or advection calculation

Expects 1 integer value; default is 362; type is *F*.

Purpose: This specifies the desired direction to be used in the calculation of the horizontal gradient (with keyword *grad*), Laplacian (with keyword *lapl*), or advection (with keyword *hadv*). Values between, and including, 0 and 360, mean that you want the component in that compass direction. A value of 361 means that you want the magnitude of the gradient, the total Laplacian, or the total advection. A value of 362 means that you want the component in the along-cross-section (left to right) direction, where the cross section position is defined by the keywords *crsa* and *crsb*. (See descriptions of keywords *crsa* and *crsb*). A value of 363 means that you want the component into the cross section. Since the Laplacian and the advection are scalars, not vectors, use of the word "component" above means that only velocity components and derivatives in the specified direction are used. Thus, if a direction is specified (*gdir*=1-360 or *gdir*=362 or *gdir*=363), this gives the second derivative in that direction if *lapl* is used, or the velocity component in that direction times the derivative of the field in that direction if *hadv* is used.

grad: Calculate (and plot) the gradient of a field

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag causes RIP to calculate (and plot) the horizontal gradient (on a height surface) of the field specified by the *feld* keyword, instead of the field itself. Depending on the setting of the keyword *gdir*, either the magnitude of the gradient will be calculated, or the component of the vector gradient in the direction specified will be calculated.

hadv: Calculate (and plot) the horizontal advection of a field

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag causes RIP to calculate (and plot) the horizontal advection (using constant-height derivatives) of the field specified by the *feld* keyword, instead of the field itself. Depending on the setting of the keyword *gdir*, either the total advection will be will be calculated, or the advection due only to the wind component in the specified direction times the derivative of the field in that direction will be calculated.

hide: Hide contours, vectors, or streamlines that are below ground.

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag causes contours, vectors, and streamlines to be omitted in a grid box if the chosen vertical level is below ground at any of the four corners of the grid box. If *hide* is not used, then for points where the chosen vertical level is below ground, pressure and geopotential height are reduced hydrostatically, some temperature-related variables are reduced with a standard lapse rate, and all other variables are assigned the surface value. Also, if *imakev5d*=1, *hide* will cause the height-interpolated data to be assigned the "missing" flag below ground, so that Vis5D will show no contours, vectors, etc. below ground. If *hide* is not used, values will be assigned to variables at below-ground points as described above.

hodo: Plot a hodograph.

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag is used in conjunction with "ptyp=sv", and has the effect of producing a hodograph in the upper left corner of the sounding plot. If a skew-T background is also drawn with "feld=tic; ptyp=sb" (which it almost always is when you create a skew-T sounding), then the *hodo* keyword should also be included on the plot specification line for the background, so that parts of the background will be omitted in the area where the hodograph is drawn.

hvbr: Specify the orientation (horizontal or vertical) of the filled contour label bar.

Expects 1 integer value; default is -1 (let RIP decide); type is *F*.

Purpose: For color-filled contour plots, RIP creates a label bar, which shows what values correspond to what colors. This bar may appear either as a horizontal bar on the bottom of the plot, or as a vertical bar on the right side of the plot. If *hvbr* is NOT used, RIP chooses which orientation is best, based on the aspect ratio of

the plot. If you want to override RIP's choice of orientation, use *hvbr*=0 for a horizontal bar on the bottom, or *hvbr*=1 for a vertical bar on the right.

incl: Insert (at run time) an "include" file of plot specification statements into the current plot specification table.

> Expects 1 character value; there is no default value; type is *F*.

> Purpose: This keyword allows the user to tell RIP to insert (at run time) additional plot specification information from another file into the plot specification table. This capability makes it easier to repeat large sections of plot specification information in a single input file, or to maintain a library of "canned" plot specifications that can be easily included in different input files. The include file is specified as *incl=filename*, where *filename* is the pathname (relative to the current working directory) of the file to be inserted. Use of this function has the effect of simply inserting the contents of the included file in place of the line on which the *incl* keyword appears. In the original line, only the *incl=filename* specification is processed. Any text prior to the *incl* keyword is ignored, and any text after it is considered to be part of the file name to be included.

intv: Vector or character interval

> Expects 1 integer value; default is 1; type is *F*.

> Purpose: This specifies the plotting stride, in grid intervals. For example, if *intv*=2, vectors or characters will be plotted at every other grid point. It also specifies a spacing of streamlines for horizontal streamline plots, though the quantitative meaning of, say, *intv*=3 for streamlines is unclear. For streamlines, it's best to experiment with values from 1-10 and see what looks nice.

lapl: Calculate (and plot) the horizontal Laplacian of a field

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: This flag causes RIP to calculate (and plot) the horizontal Laplacian (on a constant-height surface) of the field specified by the *feld* keyword, instead of the field itself. Depending on the setting of the keyword *gdir*, either the total horizontal Laplacian will be will be calculated, or the second derivative of the field in the specified direction will be calculated.

lcbr: Label color for filled contour label bar

> Expects 1 character value; default is *def.foreground*; type is *F*.

> Purpose: This sets the color of the label bar text and box perimeter. Any color name that is defined in the color table is valid.

lchl: Label color for high/low labels

> Expects 1 character value; default is the value of *colr*; type is *F*.

> Purpose: For contours, this sets the color of the high/low label text and box perimeter, if it is to be different from *colr*. For trajectories, this sets the color of the "L" marking the storm position for storm-relative trajectories. Any color name that is defined in the color table is valid.

lcll: Label color for labeled contours or trajectories.

Expects 1 character value; default is the value of *coll* for contours, or the value of *colr* for trajectories; type is *F*.

Purpose: For contours, this sets the color of the contour label text and box perimeter, if it is to be different from *coll*. For trajectories, this sets the color of the trajectory labels. Any color name that is defined in the color table is valid.

lclo: Label color for low labels in contour plots.

Expects 1 character value; default is the value of *lchl*; type is *F*.

Purpose: This sets the color of the low label text and box perimeter, if it is to be different from *lchl*. Any color name that is defined in the color table is valid.

lcnl: Label color for negative labeled contours.

Expects 1 character value; default is the value of *conl*; type is *F*.

Purpose: This sets the color of the negative contour label text and box perimeter, if it is to be different from *conl*. Any color name that is defined in the color table is valid.

lczr: Label color for zero contour.

Expects 1 character value; default is the value of *cozr*; type is *F*.

Purpose: This sets the color of the zero contour label text and box perimeter, if it is to be different from *cozr*. Any color name that is defined in the color table is valid.

levs: Level specifier for horizontal plots.

Expects an arbitrary number of real values; default is a single value equal to the index of the lowest model level; type is *R*.

Purpose: This defines the vertical levels to be plotted. The *levs* keyword has many complexities to it, which are described below.

The way in which the values of *levs* are specified depends on the setting of *vcor*:

For *vcor=s*, *levs* should be specified by vertical level index (i.e. *k* index). When *k* indices are used, you may specify them either in RIP's conventional way (assuming top-most level is *k*=1 and bottom-most level is *k*=KMAX), or in reverse orientation with a following "fb" to indicate "from the bottom" (bottom-most level is *k*=1fb, second level above ground is *k*=2fb, etc.). For *vcor=s*, RIP will not interpolate between model levels--only discrete values of k-index can be specified.

For other values of *vcor*, RIP will interpolate to the requested level.

For *vcor=p, l*, or *x* (pressure, log pressure, or Exner function), *levs* should be specified in hPa; for *vcor=z* (height), km; for *vcor=t* or *e* (potential temperature or equivalent potential temperature), K; for *vcor=m* (temperature), C; and for *vcor=q* (potential vorticity), PVU.

Any number of levels can be requested, separated by commas. For example, consider the following:

```
vcor=p; levs=1000,850,700,500;
vcor=s; levs=3,5,2fb,15,1fb,22;
vcor=z; levs=2.5,9.4,5;
vcor=t; levs=290,330,360;
```

The first example says you want plots at pressure = 1000, 850, 700, and 500 mb. The second example says you want plots at model levels, and those levels will be k=3, k=5, k=2nd level from the bottom, k=15, the bottom level, and k=22. The third example says you want plots at z = 2.5, 9.4, and 5.0 km. The fourth example says you want plots at θ = 290, 330, and 360 K.

There is a simple way to specify a range of levels without having to ask for each one individually. This is done by use of the minus sign. If the nth *levs* value has a minus sign, that means you want plots from the (n-1)th value to the (-n)th value, with an increment of the (n+1)th value. This range specification style can be intermixed with individual level values. This is all best understood by example. Consider the following examples:

```
vcor=s; levs=15,1fb,-26,2,17,21,-2fb,3,20fb
vcor=p; levs=1000,-700,150,500,400,300,-50,50
```

The first example will give plots at model levels, and (if there are 34 model levels total) those levels will be at k=15, k=34, k=32, k=30, k=28, k=26, k=17, k=21, k=24, k=27, k=30, k=33, and k=15,

The second example will give plots at pressure = 1000, 850, 700, 500, 400, 300, 250, 200, 150, 100, and 50 mb.

How does the plotting algorithm respond to PSLs that ask for more than one level? The way it works is that for a given frame, if any of the plots within the frame are requested at more than one level, then the entire frame is repeated for each of the levels requested for that plot. If more than one plot in a given frame is requested at multiple levels, then the frame will be repeated n times, where n is the maximum number of levels requested for any of the plots in the frame. The frames with less than n levels requested will have the last level repeated until n frames have been plotted. So then, consider the following unlikely, but illustrative, example of a FSG:

```
====================================================
feld=tmc; ptyp=hc; vcor=s; levs=27,-23,1
feld=wsp; ptyp=hc; vcor=p; levs=850,400,300
feld=map; ptyp=hb
feld=tic; ptyp=hb
====================================================
```

This example will cause the frame to repeat 5 times, with the following plots overlaid:

1. Temp. (C) at model level 27, Wind speed at pressure=850mb, map background, and tick marks on the plot perimeter.

2. Temp. (C) at model level 26, Wind speed at pressure=400mb, map background, and tick marks on the plot perimeter.

3. Temp. (C) at model level 25, Wind speed at pressure=300mb, map background, and tick marks on the plot perimeter.

4. Temp. (C) at model level 24, Wind speed at pressure=300mb, map background, and tick marks on the plot perimeter.

5. Temp. (C) at model level 23, Wind speed at pressure=300mb, map background, and tick marks on the plot perimeter.

Note that the value(s) of *levs* are irrelevant and ignored for background plots (maps, tick marks), vertical cross sections, soundings, or horizontal plots of 2-d fields (sea-level pressure, ground temperature, etc.).

Also note: For character plots of 3D fields, only model levels can be used (*k* indices). Interpolation to other vertical coordinates is not available for character plots (i.e. the keyword *vcor* has no effect for horizontal character plots).

Vertical averaging: the *levs* keyword can also be used to average a 3D field in the vertical. This only works with *vcor=s*, i.e. you can only average between model levels, not between pressure levels or isentropic levels, etc. The specification is similar to the multiple-level format for the *levs* keyword, except that the third number should be a zero. The following PSL is an example:

```
feld=tmc; ptyp=hc; vcor=s; levs=28,27,20,-12,0,8
```

This would plot temperature at model levels 28 and 27, then a vertical average of temperature from model levels 20 through 12, and then temperature at model level 8.

Vertical differencing: the *levs* keyword can also be used to do model-level differencing for a 3D field. As with averaging, this only works with model vertical levels as the vertical coordinate (*vcor=s*), i.e. you can only take a difference between model levels, not between pressure levels or isentropic levels, etc. The specification is similar to vertical averaging, except that the third number should be a -1. The following PSL is an example:

```
feld=tmc; ptyp=hc; vcor=s; levs=28,27,20,-12,-1,8
```

This would plot temperature at model levels 28 and 27, then a vertical difference (temperature at model level 20 minus temperature at model level 12), and then temperature at model level 8.

linw: Line width

Expects 1 integer value; default is 1; type is *F*.

Purpose: This sets the line width, as a multiple of the default line width for NCAR Graphics. Depending on plot type, the line width applies to contours, vectors, wind barbs, streamlines, trajectories, swarms, trajectory net ascent circles, lat/lon

lines, plot perimeter and tick marks, boxes or lines on horizontal plots, skew-T temperature trace, or skew-T background grid.

lwll: Labeled contour line width

Expects 1 integer value; default is the value of *linw*; type is *F*.

Purpose: This sets the line width of the labeled contours, if it is to be different from *linw*.

lwng: Negative contour line width

Expects 1 integer value; default is the value of *linw*; type is *F*.

Purpose: This sets the line width of the negative contours, if it is to be different from *linw*. Also, for horizontal trajectory plots, if *feld=circle* is set, this line width is used for circles representing negative net height changes.

lwnl: Negative labeled contour line width

Expects 1 integer value; default is the value of *lwng*; type is *F*.

Purpose: This sets the line width of the negative labeled contours, if it is to be different from *lwng*.

lwzr: Zero contour line width

Expects 1 integer value; default is the value of *linw*; type is *F*.

Purpose: This sets the line width of the zero contour, if it is to be different from *linw*.

mand: Show mandatory levels on skew-T background plot

Expects no values (logical); default is *.false.*; type is *F*

Purpose: The normal sounding background has horizontal black lines every 100 hPa, and light gray lines at the intermediate 50 hPa levels. *mand* changes this to black lines at mandatory levels, and light gray lines at all non-mandatory levels divisible by 50 hPa.

mfco: Map fill colors

Expects 1 to 6 character values; default is no color filling of map; type is *F*.

Purpose: For maps, this sets the fill colors. If only one color is specified, the entire map will be filled with that color. If two colors are specified, water areas will be filled with the first color, and land areas will be filled with the second color. If six colors are specified, for NCAR Graphics map backgrounds ('PS', CO', 'Earth..', etc.), water areas will be filled with the first color, and the different land areas (states, countries) will be filled with the other five colors, so that no adjacent land areas are the same color; for RANGS/GSHHS backgrounds, water areas will be filled with the first color, and lakes, islands, and ponds on islands will be filled with the next three colors, respectively. Filling is not available for custom map backgrounds.

mjsk: Major (labeled) contour skip increment

Expects 1 integer value; default is 3; type is *F*.

Purpose: This specifies the number of minor (unlabeled) contours between major (labeled) contours. A value of 3 means every fourth contour will be labeled.

mllm: Lat/lon line mask

Expects 1 character value; default is *none*; type is *F*.

Purpose: For maps, this keyword determines whether lat/lon lines will be masked (i.e. will NOT appear) over *land* or *water*. The default value of *none* results in no masking, in which lat/lon lines appear everywhere. Masking is only available with NCAR graphics map backgrounds, not with RANGS/GSHHS or custom map backgrounds.

mult: Multiplicative contour interval

Expects no values (logical); default is *.false.*; type is *F*

Purpose: This causes *cint* to be used as a contour interval multiplier rather than a contour interval increment. Examples will clarify this:

Example 1:

```
cbeg=3.5; cend=6.5; cint=.5
```

In example 1, *mult* is not used. Therefore, you will get contours at 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, and 6.5.

Example 2:

```
mult; cbeg=2; cend=162; cint=3
```

In example 2, since *mult* is used, you will get contours at 2, 6, 18, 54, and 162.

ncon: Nice number of contours

Expects 1 integer value; default is 20; type is *F*.

Purpose: This specifies a nice number of contours that is used by RIP as an approximate target number of contours for picking a value of *cint*, if you do not specify *cint*.

nmin: Maximum number of "model info" lines to print at the bottom of the frame

Expects 1 integer value; default is 5; type is *F*.

Purpose: Currently, the various *ripdp* programs create a ".minfo" file that contains one or more lines of information about the model run, with the first line of information being most important, and subsequent lines being of lesser importance. This parameter tells RIP how many of these lines should be printed at the bottom of the frame. The default value of 5 is probably larger than the number of lines that will ever exist, so this effectively means "print all available lines". If set to 0, no model info will be printed. *nmin* only needs to be set for one plot in the frame, and it will then apply to the entire frame.

nmin: No model info message

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: By default, RIP writes a model information line at the bottom of the frame if a *.minfo* file is available with the data set. If *nmin* is included in any of the plots specification lines for a particular frame, the model information line is omitted in that frame.

nmsg: No message

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: This flag suppresses the message that may appear at the bottom of plots, which shows units, contour intervals, maximum vectors, trajectory widths, etc.

nobr: No label bar for filled contour plots

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: This flag suppresses the drawing of the label bar

nogd: No cross-hatching underground

> Expects no values (logical); default is *.false.*; type is *F*.

> Purpose: For vertical cross sections, RIP always shows a line that represents the ground. For pressure- or height-level data sets, RIP also normally fills in the below-ground region with cross hatching, to cover up the plotted contours or vectors of fictitious data below ground. This cross-hatching can be disabled with the *nogd* keyword.

nohl: No high and/or low markers

> Expects either one value or no values (hybrid character/logical); default is *.false.*; type is *F*.

> Purpose: This flag suppresses the marking of highs and/or lows in contour plots. When used as a simple logical flag (nohl;), it suppresses both high and low labels. If set to "H" (nohl=H;), it suppreses only highs. If set to "L" (nohl=L;), it suppresses only lows.

> There are three additional values that *nohl* can take: *Z*, *T*, or *A*. These do not suppress the labeling of extrema, but change it. *nohl=Z* causes highs and lows to be labeled with "H" and "L" only, with no values shown; *nohl=T* causes highs and lows to be labeled with "W" (warm) and "C" (cold) only, with no values shown; and *nohl=A* causes highs and lows to be labeled with "A" (anticyclonic) and "C" (cyclonic) only, with no values shown.

> In trajectory plots, *nohl* should only be specified without any values, and it acts to suppress the plotting of the storm center position with an "L" when the *tjsp* keyword has been set.

nolb: No contour or trajectory labels

> Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag suppresses the labeling of contours or trajectories.

nozr: No zero contour

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag suppresses the zero contour.

nsmm: No smoothing message

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag suppresses the message (at the end of the plot title) indicating the number of smoothing passes used for this plot.

nttl: No plot title

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag suppresses the title that appears for each plot at the top of the frame.

nvlb: No vertical level message

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag suppresses the message (in the middle of the plot title) indicating the vertical level for this plot.

orlb: Orientation of contour labels.

Expects on integer value; default is 1; type is *F*.

Purpose: There are three possible values of this keyword. A value of 1 causes all contour labels to be drawn at a horizontal orientation. A value of 2 causes all labels to be oriented along the contour. A value of three uses the old CONREC method of contour labeling (instead of the CONPACK method) in which the labels are drawn with cruder character strings that are actually part of the contour dash pattern.

ouco: Map outline color

Expects 1 character value; default is the value of *colr*; type is *F*.

Purpose: For maps, this sets the color of the map outlines. Any color name that is defined in the color table is valid.

ouds: Map outline dot/solid flag

Expects 1 character value; default is *dot*; type is *F*.

Purpose: For maps, this keyword determines whether outlines will be dotted (*dot*) or solid (*solid*). Only works with NCAR graphics map backgrounds, not with RANGS/GSHHS or custom map backgrounds. Outlines are always solid with those backgrounds.

oulw: Map outline line width or dot spacing

Expects 1 integer value; default is 1 (if *ouds=solid*) or 12 (if *ouds=dot*); type is *F*.

Purpose: If *ouds=solid*, this keyword determines the map outline line width, as a multiple of the default line width for NCAR Graphics. If *ouds=dot*, this keyword determines the map outline dot spacing, as a multiple of the default dot spacing for routine EZMAP, which is 1/4096 times the width of the plotting screen.

outy: Map outline type

Expects 1 character value; default is *PS*; type is *F*.

Purpose: For maps, this keyword determines what type of outlines will be used. The traditional NCAR Graphics choices are *NO* (no outlines), *CO* (continental only), *US* (U.S. states only), *PS* (continental, international, and U.S. states), or *PO* (continental and international). The new NCAR Graphics "Earth.." outlines can also be accessed by setting *outy=Earth..nLm*, where *n* should be 1, 2, or 3, specifying the dataset, and *m* should be 1 through 5, specifying the "level" of outline desired. See the NCAR Graphics EZMAP documentation for more information. You can also access the very high-resolution RANGS/GSHHS outline data set, as implemented in NCAR Graphics, by setting *outy=RGn*, where *n* should be 0 through 4. *n* specifies the resolution desired, 0 being finest and 4 being coarsest. Again, see the NCAR Graphics EZMAP documentation for more information. Note that use of the RANGS/GSHHS outline data set is only available in NCAR Graphics version 4.3 or later. If *n* is omitted, the code will make the choice for you based on the size of the map being drawn. Finally, you can also draw custom-made map outlines by using custom map files. Several pre-made files (in either *.ascii* or *.bin* format) are available from both the MM5 ftp web site and from Mark Stoelinga's ftp site at the UW. The desired file should be placed in a directory called ${RIP_ROOT}/custom_maps (which you need to make). Once the desired file is in place, it can be accessed with the *outy* keyword. For example, if you make a file with U.S. rivers and call it ${RIP_ROOT}/custom_maps/rivers.ascii, you could draw this map background by requesting *feld=map;* and *outy=rivers.ascii;*. Functionally, there is no difference between the *.ascii* file and the *.bin* file of the same name. The *.bin* files have the same information except in binary, which makes them smaller and provides faster I/O. Several examples of map definition files are provided on the ftp site. If you want to make your own map definition file, consult the read and format statements in subroutine *hiresmap.f*, to determine the format that the map definition file should have. IMPORTANT NOTE: dotted outlines, color filling, and masking of lat/lon lines do not work if a custom map background is used.

plrs: Make sounding background more suitable for cold atmosphere

Expects no values (logical); default is .false.; type is F.

Purpose: This flag causes the sounding background to be shifted leftward to colder temperatures. This allows very cold soundings, e.g. in polar atmosphere, to be better plotted. This flag must be used with all overlays in a sounding plot [background (ptyp=sb), contour (ptyp=sc), and vector (ptyp=sv)].

pslb: Write horizontal contour plot slab data to file.

Expects no values (logical); default is .false.; type is F.

Purpose: This flag causes RIP to write the contents of the 2-D slab of data to a binary file, just prior to plotting in the horizontal contour plotting routine. The file is written to Fortran unit 59, and will appear as "fort.59" in the working directory when RIP execution is complete. The file will contain as many slabs of data as there were separate horizontal contour plot overlays in the entire RIP execution. Each slab is written in coumn-major order.

pwbr: Perimeter line width for label bar in filled contour plots

Expects 1 integer value; default is 1; type is *F*.

Purpose: This sets the perimeter line width for the label bar. A value of 0 indicates that no perimeter should be drawn around the boxes in the label bar.

pwhl: Perimeter line width for high/low labels in contour plots

Expects 1 integer value; default is the value of *pwlb*; type is *F*.

Purpose: This sets the perimeter line width for high/low labels, if it is to be different from *pwlb*.

pwlb: Perimeter line width for contour labels

Expects 1 integer value; default is 1; type is *F*.

Purpose: This sets the perimeter line width for labels, as a multiple of the default line width for NCAR Graphics.

qgsm: Smoothing, as an intermediate step in quasigeostrophic and Sawyer-Eliassen diagnostics.

Expects 1 integer value; default is 0; type is *F*.

Purpose: This specifies the number of smoothing passes to be applied to various fields that go into the quasigeostrophic and Sawyer-Eliassen diagnostic routines. See the description of the fields *qgomg* and *qmomg*. There are several types of smoothers available. The specification of the type and severity of smoothing is the same as for the plotting smoother keyword *smth*. See the description of keyword *smth* for more information.

redo: Recalculate the field even if it is available in a file.

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: Sometimes, a diagnostic field may have already been calculated and saved to a file. RIP first checks to see if this is the case, and, if so, reads in the file instead of recalculating the field. However, occasionally you may want RIP to recalculate the field anyway. This flag forces RIP to do that.

rfst: Reference state

Expects 4 real values; default is 1000, 290, 50, 0.1; type is *F*.

Purpose: This specifies a reference state (as a function of height), and is used only for computing the plotted fields of perturbation temperature (*tpt*) and perturbation pressure (*ppt*). The reference state should be specified by four values: sea-level pressure (mb), sea-level temperature (K), lapse rate (*dT/d[ln(p)]*, K), and

stratospheric temperature (K). The default values correspond roughly to the U.S. Standard Atmosphere, with no stratospheric isothermal layer because stratospheric temperature is set to 0.1 degree above absolute zero.

rota: Rotate the domain by increments of 90 degrees

Expects 1 real values; default is 0.; type is *R*.

Purpose: This causes the domain to rotate counterclockwise by the angle specified in degrees. Possible values are -90, 0, 90, and 180. Only works for horizontal plot types, and for data sets with polar stereographic projection.

save: Save the specified diagnostic field(s) to a file.

Expects no values (logical), though one can be provided (character); default is *.false.*; type is *F*.

Purpose: Sometimes, a diagnostic field may take some time to calculate. It is therefore some times desirable to save that field to a file, so that upon subsequent requests for that field, it is simply read in rather than recalculated. The *save* keyword causes RIP to save the specified field(s) to a file. The file name will follow the same format as the standard RIP data files. The last part of the file name will be the same as the requested field (e.g., *myrun_024.00000_cap3* if *feld=cap3* was used). However, it is also possible to explicitly assign the last part of the file name by setting the *save* keyword equal to a character string. This may be useful if the field was altered by one of the keywords that affects a field after it is calculated (e.g., *addf, smcp, grad, lapl, hadv, diff,* etc.)

sepa: Sawyer-Eliassen diagnosis parameters.

Expects 9 real values; defaults are as mentioned below; type is *R*.

Purpose: This defines several parameters that are used for solving the Sawyer-Eliassen equation. Nine different parameters must be set, though they each have a default value that is used if that particular parameter is not set. The parameters are as follows:

The first five parameters are relevant for either the moist or dry S-E solver:

1. Maximum number of iterations in the over-relaxation (default is 200)
2. Minimum residual required for convergence (default is 1.0 m mb s^{-1})
3. Over-relaxation factor (default value is 1.6)
4. Averaging distance in the along-front (cross-section-normal) direction, in grid spaces. This is similar to the *xavg* keyword that can be used for other cross section plots; however, this parameter causes averaging to be performed on fields from which the Sawyer-Eliassen equation terms are calculated, rather than on the final field being plotted. The default value is 0 (no averaging).
5. Horizontal smoothing factor. This allows for horizontal smoothing of the fields (interpolated to pressure levels) from which the Sawyer-Eliassen equation terms are calculated. The smoothing factor is specified as a distance in grid spaces, and represents the width of a simple triangle-shaped weighting function for the smoother. The default is 0.0, or no smoothing.

The last three parameters are relevant only for the moist S-E solver:

6. The maximum number of "big" iterations allowed for "big" iteration convergence. "Big" iteration convergence occurs when all points have the same sign of vertical velocity for two consecutive "big" iterations. Default is 15.
7. The threshold relative humidity (as a percent value) used to determine the type of stability (moist or dry) used at a grid point. Default is 80.
8. Smoothing factor for effective stability. This is similar to the horizontal smoothing factor described above, except it applies only to the effective stability that is obtained after the moist or dry stability is chosen at each grid point. Its purpose is to remove large stability gradients across the zero vertical velocity line or the threshold relative humidity line. Default is 2.
9. Switch for SG (1) or QG (2) version of left hand side of S-E equation. Default is SG (1).

If all the defaults are satisfactory, *sepa* need not be specified at all. However, you will probably want to experiment with one or more of the parameters. If this is the case, not all nine parameters need be specified, but 8 commas must always be included, in order to define which parameters are being changed. For example, if you want to change the minimum error for convergence to 5.0, the horizontal smoothing factor to 2.5, and the stability smoothing factor to 3.0, you would specify

```
sepa=,5,,,2.5,,,3,
```

sids: Station IDs

Expects an arbitrary number of character values; Default is no station IDs; type is *F*.

Purpose: For horizontal background plot *feld=sids*, which shows requested station locations by plotting the requested station IDs at the correct locations on the map background, the keyword *sids* is used to pick the station IDs as 4-character upper-case strings. Any number of station IDs can be requested, separated by commas. Any station ID that is available in the *stationlist* file can be used.

sloc: Sounding location

Expects a variety of values; defaults are 3,3; type is *R*.

Purpose: This sets the location (in the dot point domain) of either a skew-T sounding or of a vertical profile plot. For an explanation of location settings, see the description of *crsa*.

smcp: Number of constant-pressure smoothing passes.

Expects 1 integer value; default is 0; type is *F*.

Purpose: This specifies the number of constant-pressure smoothing passes. It works the same as *smth* (see below), except that the data are first interpolated to pressure levels, then the smoothing is carried out, and then the data are interpolated back to model levels. Note: you cannot use *smth* and *smcp* at the same time, you must choose one or the other (or neither) for any given PSL.

smth: Number of smoothing passes.

Expects 1 integer value; default is 0; type is *F*.

Purpose: This specifies the number of smoothing passes. There are several types of smoothers available:

If *n* is between 1 and 99, then a 9-point weighted smoother is applied *n* times. The smoother follows equation 11-107 in Haltiner and Williams. One pass completely removes $2\delta x$ waves on the interior. On the outer row and column, and near missing data points, smoothing is carried out in a manner that preserves the domain average value of the field.

If *n* is between 101 and 199, then a smoother-desmoother is applied ($n - 100$) times. One pass removes a large fraction of the $2\delta x$ component, but is not as harsh on longer wavelengths as the 9-point smoother

If *n* is between 201 and 299, then the smoother-desmoother is applied ($n - 200$) times, and, after each pass, the data field is forced to be non-negative.

If *n* is between 301 and 399, then a weighted smoother is applied, in which the smoothed value is given by a weighted average of values at surrounding grid points. The weighting function is the Cressman weighting function:

$$w = \frac{(D^2 - d^2)}{(D^2 + d^2)}$$

In the above, *d* is the distance (in grid increments) of the neighboring point to the smoothing point, and *D* is the radius of influence [in grid increments, given by ($n - 300$)].

If *n* is between 401 and 499, then the smoothing is similar for *n* = 301-399, except the weighting function is the circular aperture diffraction function (following a suggestion of Barnes et al. 1996):

$$w = \frac{J_1(3.8317 \times d / D)}{(3.8317 \times d / D)}$$

If *n* is between 501 and 599, then the smoothing is similar for *n* = 301-399, except the weighting function is the product of the rectangular aperture diffraction function in the *x* and *y* directions (the function used in Barnes et al. 1996):

$$w = \frac{\sin(\pi x / D)}{(\pi x / D)} \times \frac{\sin(\pi y / D)}{(\pi y / D)}$$

sndg: Print sounding parameters.

Expects no values (logical); default is *.false.*; type is *F*.

Purpose: This flag causes a variety of sounding parameters to be calculated and printed out in the lower left corner of a skew-T sounding. It must be used with *ptyp=sv*! It will not work with *ptyp=sc*! Also, if a skew-T background is drawn on a sounding for which *sndg* was used with a sounding vector (*ptyp=sv*) plot, *sndg* should also be used on the plot specification line for the sounding background, so that RIP will omit drawing skew-T background lines in the same area as that where the sounding parameters are printed.

strm: Storm velocity

Expects 1 or 2 real values; default is 0.0; type is *F*.

Purpose: The storm velocity is used to plot storm-relative wind vectors, speeds, or trajectories. The storm velocity can be specified in 2 ways. The first way is as a speed which is assumed to be in the along-cross-section (left to right) direction, where the cross section position is defined by the keywords *crsa* and *crsb*. (See descriptions of keywords *crsa* and *crsb*.) The second way is by giving two values for *strm*, separated by a comma, which are the *x* and *y* components of the storm velocity. In either case, the *x* and/or *y* components of the storm motion will be subtracted from the *x* and/or *y* components of the wind prior to plotting. This keyword only effects the following fields: *uuu, vvv, ugeo, vgeo, xptgeo, xntgeo, bvfsq, spsq, wsp, wspk, wdir, xnt, amt,* and *xpt*; and the five trajectory fields *arrow, ribbon, swarm, gridswarm.* and *circle,*

titl: Plot title

Expects 1 character string; default is that RIP automatically generates the tile for each plot overlay; type is *F*.

Purpose: Normally, RIP automatically generates a title for each plot overlay at the top of the frame underneath the frame title. You can specify your own title for a plot overlay using *titl*. Because blank characters are not recognized in the plot specification table, you must use the underscore character to indicate that you want a blank character at a particular position in your specified plot title. *titl* also has another special use: it can be used to print any message at any location on the plot by using it in conjunction with *feld=bull*, and it can be used to place a label above a vertical bar (*feld=vbar*).

time: Time discriminator

Expects an arbitrary number of real values; default is all times in the *ptimes* (or *iptimes*) array; type is *F*.

Purpose: This is a special keyword that should only appear by itself in a PSL. It does not apply to a single plot, but rather to the entire FSG that it is in. If you want a particular frame to be plotted only at some of the times specified in the *ptimes* (or *iptimes*) array in the namelist, then you can include the following line in the FSG for that frame:

```
time=time1,time2,time3,...
```

where each of *time1, time2, time3*, etc., are times, specified as real numbers. Like the values given for the *ptimes* (or *iptimes*) array in the namelist, the values of

time should be in hours, but do not need to be whole numbers. If a *time= PSL* is included in the FSG, then RIP will only draw that frame for those model output times that are within *tacc* seconds of one of the times in the *ptimes* (or *iptimes*) array AND are within *tacc* seconds of one of the times in the *time=* setting.

tjar: Arrow widths for trajectory plots

Expects 1 or 2 real values; defaults are .003 and .035; type is *F*.

Purpose: *arrow* and *ribbon* trajectories in horizontal plots use variable-width arrowheads, where the width of the arrowhead indicates the height of the trajectory. The two values of *tjar* specify the widths (as a fraction of the total screen width) that correspond to the lowest and highest (height-wise) values, respectively, of the vertical window, as specified by *vwin*. For the *feld=circle* trajectories, only one value is expected for *tjar*, and it represents the radius of a circle corresponding to the reference net ascent specified by *vwin*. For trajectories in a cross section (vertical trajectory plots), there is no variable width feature, so only one value is expected for *tjar*; if no value is specified, .035 is the default.

tjen: End time (in forecast hour) of plotted trajectories.

Expects 1 real value; default is the final time of the calculated trajectory; type is *F*.

Purpose: This specifies the last time (i.e. the time corresponding to the head) of the plotted trajectory. For *feld=circle* trajectories, this specifies the end time for the period during which net ascent is calculated. Note that *tjen* may be less than or equal to the actual end time of the calculated trajectory.

tjfl: File name of trajectory position info.

Expects 1 character value; default is *junk* (i.e. a useless value); type is *F*.

Purpose: This specifies the name of the file containing trajectory position information for the trajectories you want to plot. This file should already exit, since it should have already been created by a previous RIP run in trajectory calculation mode.

tjid: ID numbers of trajectories to be plotted.

Expects an arbitrary number of real values; default is a single value of 1.0; type is *F*.

Purpose: This specifies the ID numbers of the desired trajectories. For the trajectory fields *arrow, ribbon, swarm,* or *circle*, all desired trajectory IDs can be listed, or IDs can be requested in a manner similar to the way the *levs* keyword is set for requesting vertical levels. See the description of the keyword *levs* for more details. For example, a setting of

```
tjid=1,3,8,9,-27,6,32;
```

would result in the plotting of trajectory numbers 1,3,8,9,15,21,27, and 32. For a trajectory *swarm*, the desired sequence of trajectory IDs for the swarm curve should be specified in the order desired, and if a closed polygon is desired, the

first and last ID requested should be the same. For a trajectory *gridswarm*, it is assumed that part or all of the trajectories in the position file were initially arranged in a row-oriented 2-D array. In this case, instead of individually specifying each trajectory in the swarm sequence (as is done for *feld=swarm*), three values must be provided for the *tjid* keyword: (1) the ID number of the first trajectory of the gridswarm; (2) the number of columns in the gridswarm array (i.e., the direction that varies first); and (3) the number of rows in the gridswarm array (i.e., the direction that varies second).

tjsp: Storm position for storm-relative trajectories.

Expects an arbitrary number of real values, in sets of three; default is such that this keyword is not set to anything, disabling its effect; type is *F*.

Purpose: Trajectories can be plotted in a storm-relative sense. In order to do this, RIP needs to know where the storm is. This information is provided through either the *tjsp* keyword or the *strm* keyword. If the storm motion can be approximated by a constant speed and direction, then using the *strm* keyword is easiest. See the explanation of keyword *strm*. If the storm motion is more complicated, then *tjsp* must be used. Each set of three values is a time (in model forecast hours), an *x* value, and a *y* value. Use as many sets as you need to cover the period of interest and to sufficiently resolve the storm motion. RIP linearly interpolates in time and space to get the storm position between the provided times. An IMPORTANT rule is that the *x* and *y* values must be relative to the coarsest domain grid (the domain which is centered on the central lat. and lon.). This is to accommodate the possibility that the data set you are working with is from a moving nest. When storm-relative trajectories are plotted (by either method) against other fields or a map background, the only point along the trajectory that is actually where it appears to be (relative to those other fields) is the point that corresponds to the time being plotted. If *tjsp* is used, RIP marks the storm position at the time being plotted with an "L", although this feature can be disabled with the keyword *nohl*.

tjst: Start time (in forecast hour) of plotted trajectories.

Expects 1 real value; default is the beginning time of the calculated trajectory; type is *F*.

Purpose: This specifies the first time (i.e. the time corresponding to the tail) of the plotted trajectory. For *feld=circle* trajectories, this specifies the beginning time for the period during which net ascent is calculated. Note that *tjst* may be greater than or equal to the actual start time of the calculated trajectory.

tjti: Time interval for plotted trajectories, in hours.

Expects 1 real value; default is 1 hour; type is *F*.

Purpose: This specifies the time interval between arrow heads for arrow or ribbon trajectories, or between swarms for swarm or gridswarm trajectories. Note that this is typically greater than the time step used to calculate the trajectories.

tshl: Text size for high/low labels.

Expects 1 real value; default is the value of *tslb*; type is *F*.

Purpose: In contour plots, this sets the text size for high/low labels, if it is to be different from *tslb*. For trajectories, this sets the text size of the "L" marking the storm position for storm-relative trajectories.

tslb: Text size for labels

Expects 1 real value; default is .01; type is *F*.

Purpose: This sets the text size (i.e. character width) for labels, as a fraction of the width of the full plotting screen. It applies to contour labels, lat/lon labels (in maps), perimeter tick mark labels, and trajectory labels. It also sets the size of the bullet (or text) drawn when *feld=bullet*, and it sets the size of the label for *feld=vbar*.

tynt: Typhoon track symbol interval (hours)

Used with feld=track. Default is every forecast output time. A typhoon symbol (and label) is plotted every tynt hours. Increasing tynt may improve legibility. *tynt* does not affect the track printout.

v5nm: Variable name for Vis5D

Expects 1 character string; default is whatever was specified for *feld*; type is *F*.

Purpose: In Vis5D data set preparation mode (i.e., if *imakev5d*=1 in the *&userin* namelist), RIP assigns to each requested variable a name that will identify that variable when Vis5D is run. For the velocity fields, the Vis5D names are always *U*, *V*, and *W*. For all other variables, the name is the same as what was requested with the *feld* keyword. However, in some cases, you may want to assign a different name. Use *v5nm* to do this. This keyword should only be used when RIP is run in Vis5D data set preparation mode.

vcmx: Maximum vector value

Expects 1 real value; default is 0.; type is *F*.

Purpose: This specifies the magnitude of a vector that exactly reaches the tail of the next adjacent vector. It should be specified in the same units as those of the vector component fields, typically m s^{-1}. If *vcmx* is set to zero, then RIP will choose the maximum vector magnitude in the data as the magnitude of a vector that reaches the tail of the next adjacent vector.

Special instructions for different plot types:

ptyp=hv or *vw*: If *vcmx* is specified as any negative number, then the vectors will be plotted with the barb/flag convention.

ptyp=sv: these vectors can only be plotted with the barb/flag convention, so *vcmx* is irrelevant.

ptyp=vv: the "next adjacent vector" means the next adjacent vector in the horizontal direction, and *vcmx* should be specified in units of the horizontal components, which will probably be different from the units of the vertical component. Some experimentation may be required to get a feel for this keyword

in vertical vector mode. These vectors can only be plotted as arrows, not as wind barbs, because the units in the horizontal and vertical directions are different.

vcor: Vertical coordinate

Expects 1 character value of length 1; default is s; type is *R*.

Purpose: This defines the vertical coordinate to be used for (1) determining plotting levels in horizontal plots; or (2) defining the vertical axis in cross section plots; or (3) defining the variable width feature of *arrow, ribbon* or *circle* trajectories drawn in horizontal plots. The possible choices for the different types of plots are:

Horizontal plots or trajectory initial points:

s: model vertical level (*k*) index
p: pressure
z: height
t: potential temperature
e: equivalent potential temperature
m: temperature
q: potential vorticity

Vertical cross sections (excluding *ptyp=vv*) or horizontal trajectory plots:

s: model vertical level index
p: pressure
x: Exner function (as in pseudoadiabatic charts)
l: log pressure
z: height
t: potential temperature
e: equivalent potential temperature
m: temperature
q: potential vorticity

Vectors in the plane of a cross section (*ptyp=vv*):

s: model vertical level index
p: pressure
z: height

(These are the only choices for *ptyp=vv* because they are the only vertical coordinates for which RIP can calculate an appropriate vertical velocity).

vvms: Minimum pressure difference between plotted vectors for cross sections and soundings

Expects 1 real value; default is 0.0; type is *F*.

Purpose: This specifies the minimum pressure increment between two vectors in a column. Due to the fact that the data are not interpolated to a regular grid in the vertical, the vectors can become crowded in the high-resolution PBL. If this keyword is set to some non-zero value, for example 30, then RIP will not plot a vector if it is less than 30 hPa away from the vector below it.

vvnx: Number of vectors in the *x*-direction

Expects 1 integer value; default is 20; type is *F*.

Purpose: This specifies an approximate number of vectors that are desired in the horizontal direction. It is approximate because of the way RIP sets up the cross section grid.

vwin: Vertical window.

Expects 1 or 2 real values; default is as shown below; type is *R*.

Purpose: This determines the vertical window to use for cross sections, or for the variable width feature of trajectory arrows, ribbons, or circles in a horizontal plot. The first value is the bottom of the window, and the second is the top. For trajectory circles, only one value is specified, and this determines the reference vertical displacement amount that corresponds to a circle of diameter set by *tjar*. The units of the given values are assumed to be as follows:

vcor	Units	Default values for vertical cross sections
's'	Dimensionless	1.0, 0.0
'p','l', or 'x'	Mb	1050, 100
'z'	Km	0.0, 15.0
't' or 'e'	K	260, 400
'm'	C	-60, 40
'q'	PVU	-0.5, +5.5

For horizontal trajectory plots, the default values are the minimum and maximum values of the trajectories being plotted during the time interval set by *tjst* and *tjen*. For *feld=circle* trajectory plots, only one value of *vwin* is expected, and it specifies the net ascent that corresponds to the circle radius specified by *tjar*; if no value is specified, the maximum absolute net ascent for all the trajectories requested is the default value.

wdbr: XXXXX Distance between label bar and plot for filled contour plots

Expects 1 real value; default is -1; type is *F*.

Purpose: For color-filled contour plots, RIP creates a label bar which shows what values correspond to what colors. *wdbr* sets the width of the area used for the label bar (including labels), as a fraction of the total width of the screen. The default value of -1 tells RIP to try to optimize the width. This is usually sufficient for typical plots, i.e. it is large enough to allow room for tick labels and lat/lon labels. However, in some cases, you may want to change this width, if various messages or labels from other plots are being covered up by the label bar.

xavg: cross-section averaging distance

Expects 1 integer value; default is 0; type is *F*.

Purpose: Rather than plotting the value of a variable at a point on the cross section, you may want to plot an average along a line that extends into and out of the cross section. This may be useful for getting the average sense of quasi-2D features such as fronts or squall lines. The averaging distance into and out of the cross section is given by *xavg* in units of grid points. Hence, if the grid space is 40 km, and *xavg* is specified as 2, each point plotted on the cross section would represent an average from 80 km out of the cross section to 80 km into the cross section, or a total of 160 km.

xwin: Plotting window in the x direction

Expects 2 integer values; defaults are 1, *mjx* (*mjx* is the number of dot points in the *x* (or *j*) direction); type is *R*.

Purpose: This sets the left and right limits of the desired subdomain for a horizontal plot, specified as grid point values on the dot-point domain. If the first value given is outside the range 1 to *mjx*, it is changed to 1. If the second value given is outside the range 1 to *mjx*, it is changed to *mjx*. This can also be used to save a reduced-size domain when creating a Vis5D data set (*imakev5d*=1), in which case it should be specified only once, with the first requested variable.

ywin: Plotting window in the y direction

Expects 2 integer values; defaults are 1, *miy* (*miy* is the number of dot points in the *y* (or *i*) direction); type is *R*.

Purpose: This sets the bottom and top limits of the desired subdomain for a horizontal plot, specified as grid point values on the dot-point domain. If the first value given is outside the range 1 to *miy*, it is changed to 1. If the second value given is outside the range 1 to *miy*, it is changed to *miy*. This can also be used to save a reduced-size domain when creating a Vis5D data set (*imakev5d*=1), in which case it should be specified only once, with the first requested variable.

Appendix B. Available fields for plotting

This appendix is a list of all fields that are currently available to be plotted in RIP, i.e. the names that can be used for the *feld* keyword. For each field, the name is given, the field is described, the units are given, and the dimensionality of the field (2D or 3D) is given. Capital letters in the field name indicate a variable character for which there is more than one choice.

a. Background fields (units and dimensionality are irrelevant):

map: Map background.

> This field results in a map background being drawn in the frame. Works only if *ptyp=hb*.

box: Box.

> This field results in a box being drawn in the frame, the size and location of which is determined by the keywords *crsa* and *crsb*. Works only if *ptyp=hb*.

line: Line.

> This field results in a line being drawn in the frame, the size and location of which is determined by the keywords *crsa* and *crsb*. Works only if *ptyp=hb*.

bullet: Bullet.

> This field results in the drawing of a bullet (i.e., a solid circle), the location of which is determined by the keyword *crsa*. It can also be used to write any text string on the plot at the location specified by *crsa*. This is accomplished by setting the *titl* keyword equal to the desired text string. Use the underscore character to represent blank characters in the text string. Works only if *ptyp=hb*.

sids: Station IDs.

> This field results in the labeling of a station or stations on the plot. Use the *sids* keyword in conjunction with *feld=sids* to list the station(s) you want to be shown on the plot. See the description of *sids* in Appendix A for more information. Works only if *ptyp=hb*.

tic: Tick mark background.

> This field results in the drawing of helpful markings on the plot. It works in conjunction with the following *ptyp* settings:

> *ptyp=hb*: The perimeter is marked on all four sides with tick marks and labels denoting grid points.
> *ptyp=vb*: The left and right perimeters are marked with tick marks and labels denoting vertical coordinate increments. The bottom and top perimeters are marked with tick marks and labels denoting distance along the cross section.
> *ptyp=sb*: A standard skew-T grid is drawn and labeled, including pressure levels, temperature contours, adiabats, pseudoadiabats, and mixing ratio contours.

vbar: Vertical bar.

> This field results in the drawing of a vertical bar in a vertical cross section. The location of the vertical bar is specified with the *sloc* keyword, and it is drawn where that location projects perpendicularly onto the plane of the vertical cross section defined by the current setting of *crsa* and *crsb*. A label can be placed above the vertical bar by using the keyword *titl*. Works only if *ptyp=vb*.

b. Model output and diagnostic fields (listed in alphabetical order):

amt: Absolute momentum, m s^{-1}. (3D)

> Absolute momentum is defined here as the horizontal wind component normal to, and out of, the cross section (as defined by the *crsa* and *crsb* keywords) minus *f* (at the middle of the cross section), all times the left-to-right distance along the cross-section.

amtg: Geostrophic absolute momentum, m s^{-1}. (3D)

> Same as *amt*, but using geostrophic wind instead of the actual wind.

arrow: Trajectory arrows (see Chapter 6 for more details).

avo: Absolute vorticity (vertical component), 10^{-5} s^{-1}. (3D)

bocX: Surface precipitation type probability, percent. (2D)

> The precipitation type probability is calculated according to the REEP method of Bocchieri (1980). It uses the sounding of temperature and wet bulb temperature to determine the probabilities. Replace *X* with *l* for probability of liquid precipitation, *f* for probability of freezing precipitation, or *i* for probability of ice (or frozen) precipitation.

bshXU: Horizontal bulk wind shear with variable units depending on the value of *U*. (2D)

> The bulk wind shear is defined between the surface and up to 9 km ($X = 1$ through 9). If *X* is not specified, it defaults to shear between the surface and 6 km. Units of *bsh* can be m s^{-1} (e.g. *bsh5*), or knots (e.g. *bsh5k*).

bvfsqP: Dry Brunt-Vaisala frequency, s^{-2}. (3D)

brnshr: Bulk Richardson number shear as in Stensrud et al. (1997), m^{2}s^{-2}. (2D)

bvfsqP: Dry Brunt-Vaisala frequency, s^{-1}. (3D)

> *P* determines the inclusion of moist physics in the calculation of the B-V frequency: *d* for dry, *l* for liquid-only latent heating/cooling considerations, and *i* for liquid and ice latent heating/cooling considerations.

cap3: Convective available potential energy (CAPE) of parcels at all 3D grid points, J kg^{-1}. (3D)

> In this diagnostic field, CAPE is calculated for every grid point in the entire 3D domain, based on the lifting of a parcel starting from that grid point. It is defined as the accumulated buoyant energy from the LFC to the equilibrium level. To get CAPE as a 2D field, showing CAPE only from the parcel in each column with maximum θ_e below 3000 m AGL, use *mcap*.

cat: Clear-air turbulence index, s^{-2}. (3D)

> This index is based on Ellrod and Knapp (1992, *W&F*, March issue).

cin3: Convective inhibition (CI) of parcels at all 3D grid points, J kg^{-1}. (3D)

> In this diagnostic field, CI is calculated for every grid point in the entire 3D domain, based on the lifting of a parcel starting from that grid point. It is defined as the accumulated negative buoyant energy from the parcel starting point to the LFC. To get CI as a 2D field, showing CI only from the parcel in each column with maximum θ_e below 3000 m AGL, use *mcin*.

circle: Net ascent circles (a type of trajectory representation, see Chapter 6 for more details).

clgX: Cloud ceiling, m. (2D)

> Regardless of whether or not mixed phase microphysics were used, there are three possibilities to choose from for the determination of the phase of cloud and precipitation hydrometeors: Replace *X* with *d* for the 0 Celsius assumption, *b* for the Bocchieri assumption, or *r* for the mixed phase assumption. See code for details of these assumptions.

condheat, condheati: Diagnosed condensational heating, K h^{-1}. (3D)

> *condheat* calculates the heating due to condensation in regions of explicitly resolved saturated ascent. *condheati* does the same, except it adds in the latent heat of fusion in regions that are below freezing.

cor: Coriolis factor, s^{-1}. (2D)

ctt: Cloud-top temperature, C. (2D)

> Note: cloud-top temperature is calculated by interpolating the temperature to the level of unit optical depth into the cloud (starting at the model top). Expressions for absorption cross-section for cloud ice and cloud water are obtained from Dudhia (1989).

dbzXXXY: Simulated equivalent radar reflectivity factor, dBZ. (3D)

> This field calculates a simulated equivalent radar reflectivity factor, based on the mixing ratios of rain, snow, and graupel (*if available*). The formulas that relate mixing ratios of rain, snow, and graupel to reflectivity factor were derived assuming spherical particles of constant density, with exponential size distributions (*consistent with what is used in MM5's Reisner-2 bulk microphysical scheme*). If *X* is set to "0" (*or omitted*), then the simulated reflectivity is calculated using constant values of the intercept parameters for the size distributions of rain, snow, and graupel, as in early versions of Reisner-2 (*this is equivalent to the field dbzc used in older RIP4 versions*). If *X* is set to "1", then variable intercepts are used, as in more recent versions of Reisner-2 (*this is equivalent to the field dbzv used in older RIP4 versions*). IF *Y* is set to "1" (*0 is default*), frozen particles that are at a temperature above freezing are assumed to scatter as a liquid particle. *See comments in routine "dbzcalc.f" for more details.*

div: Divergence (of horizontal wind), s^{-1}. (3D)

dmap: Map factor on dot points, dimensionless. (2D)

dpbhAAABBB: Pressure difference between height levels, hPa. (2D)

This field is similar to thickness, except it is the pressure difference between height surfaces, rather than the height difference between pressure surfaces. The two height levels are specified as three-digit integers in units of hectometers (hm). For example, *feld=dpbh015120* would give the pressure at 1.5 km AMSL minus the pressure at 12.0 km AMSL.

dthtedz: Convective (or potential) instability, expressed as the partial derivative of θ_e with respect to height, K km^{-1}. (3D)

ehi: Energy-helicity index, dimensionless. (3D)

eth: Equivalent potential temperature, K. (3D)

ethmx: Maximum value of equivalent potential temperature below 3000 m AGL, K. (2D)

extXY: Surface extinction coefficient due to hydrometeors, km^{-1}. (2D)

Regardless of whether or not mixed phase microphysics were used, there are three possibilities to choose from for the determination of the phase of cloud and precipitation hydrometeors: Replace *X* with *d* for the 0 Celsius assumption, *b* for the Bocchieri assumption, or *r* for the mixed phase assumption. See code for details of these assumptions. The extinction coefficient is calculated for only one of the four hydrometeor types at a time: Replace *Y* with *c* for extinction due to cloud water, *r* for extinction due to rain, *i* for extinction due to cloud ice, or *s* for extinction due to snow.

fregWX: Flight regulation. (2D)

This field will assign a number from 1 to 4 to each grid point. It should therefore be plotted with the setting of *ptyp=hh*. The numbers correspond to the following flight regulation categories:

1: Visual Flight Regulation (VFR)
2: Marginal Visual Flight Regulation (MVFR)
3: Instrument Flight Regulation (IFR)
4: Low Instrument Flight Regulation (LIFR)

It can be calculated based on model-diagnosed ceiling, visibility, or both. Replace *W* with *c* for ceiling only, *v* for visibility only, or *b* for both.

Regardless of whether or not mixed phase microphysics were used, there are three possibilities to choose from for the determination of the phase of cloud and precipitation hydrometeors: Replace *X* with d for the 0 Celsius assumption, *b* for the Bocchieri assumption, or *r* for the mixed phase assumption. See code for details of these assumptions.

frgmXXX: Frontogenesis, Miller form, K (100 km h)$^{-1}$. (3D)

The Miller form of frontogenesis is the Lagrangian time rate of change of the absolute value of the horizontal gradient of potential temperature. *XXX* specifies which term on the RHS of the Miller frontogenesis equation is desired:

div: divergence term
def: deformation term
dia: diabatic term
til: titling term

The diabatic term uses a diagnosis of condensational heating assuming maintenance of 100% RH with respect to liquid water in regions of explicitly resolved saturated ascent. It does not include evaporative, sublimational, or melt cooling; fusional or depositional heating; or any diabatic results of either the cumulus or PBL parameterization schemes.

You can also request the horizontal or vertical advective terms (i.e., the advection of the absolute value of the horizontal gradient of potential temperature) by specifying *XXX* as either *had* or *vad*.

frg2dXXX: Frontogenesis, 2D frontal zone form, K (100 km h)$^{-1}$. (3D)

The 2D form of frontogenesis examines the Lagrangian time rate of change of the along-cross-section component of the horizontal gradient of potential temperature. Regardless of whether you want to display the result in a cross section or in a horizontal plot, use *crsa* and *crsb* to define the endpoints of a cross section (and, thus, the along-cross-section direction). *XXX* specifies which term on the RHS of the uni-directional frontogenesis equation is desired:

cnf: confluence term
shr: shear term
dia: diabatic term
til: titling term

The diabatic term uses a diagnosis of condensational heating assuming maintenance of 100% RH with respect to liquid water in regions of explicitly resolved saturated ascent. It does not include evaporative, sublimational, or melt cooling; fusional or depositional heating; or any diabatic results of either the cumulus or PBL parameterization schemes.

You can also request the advective terms (i.e., the advection of the along-cross-section component of the horizontal gradient of potential temperature) by specifying *XXX* as either *aad*, *iad*, or or *vad*. These refer to the advection along the cross section, the advection into the cross section, and the vertical advection, respectively. For the two horizontal advective terms, you can specify a wind speed and direction with the *strm* keyword so that relative winds will be used in the advective calculation.

Finally, all of the fields that are used to calculate the above terms can be smoothed on constant-pressure surfaces prior to the computing of products in the nonlinear terms, by using the *qgsm* keyword.

ght: Geopotential height above mean sea level, m. (3D)

ghtagl: Geopotential height above ground level, m. (3D)

gridswarm: Trajectory gridswarm (see Chapter 6 for more details).

intcld: Column-integrated cloud hydrometeors, mm. (2D)

> This is the column-integrated value of the field *qcl* (i.e., the sum of the fields *qcl* and *qci*).

intclq: Column-integrated cloud liquid water, mm. (2D)

> This is the column-integrated value of the field *qcw*.

intpcp: Column-integrated precipitation mass, mm. (2D)

> This is the column-integrated value of the field *qpr* (i.e., the sum of the fields *qra*, *qsn*, and *qgr*).

k-index: *k*-index of the grid points, dimensionless. (3D)

> This field can be used to show where the model levels are in a vertical cross section, or in a horizontal plot interpolated to an alternate vertical coordinate.

lcl: Lifted condensation level (LCL), m (AGL). (2D)

> This is the LCL for the parcel in each column with maximum θ_e below 3000 m AGL.

lfc: Level of free convection (LFC), m (AGL). (2D)

> This is the LFC for the parcel in each column with maximum θ_e below 3000 m AGL.

maxdbzX: Maximum simulated equivalent radar reflectivity factor in a column, dBZ. (2D)

> The definition of simulated equivalent radar reflectivity factor and *X* are the same as for *dbzX*.

mcap: Maximum convective available potential energy (MCAPE), J kg^{-1}. (2D)

> This is the CAPE for the parcel in each column with maximum θ_e below 3000 m AGL.

mcin: Convective inhibition (CI) for MCAPE parcel, J kg^{-1}. (2D)

> This is the CI for the parcel in each column with maximum θ_e below 3000 m AGL.

omg: Omega (*dp/dt*), μ bar s^{-1}. (3D)

pcptw: Precipitable total water, mm. (2D)

> This is the column-integrated total water (vapor plus all hydrometeors).

pcpwv: Precipitable water vapor, mm. (2D)

> This is the column-integrated water vapor only.

phydpNN: Hydrostatic pressure perturbation, mb. (3D)

> This is the same as *ppt* (below), except it is the perturbation of the hydrostatic pressure (not the total pressure) with respect to the reference state, as defined by the keyword *rfst*. A model vertical level must be chosen at which hydrostatic pressure is defined to be equal to full pressure. This level is specified as a *k*-index with *NN*

ppt: Pressure perturbation, mb. (3D)

> The perturbation is with respect to the reference state, as defined by the keyword *rfst*.

prs: Pressure, mb. (3D)

pvm: Moist potential vorticity (hydrostatic form), PVU. (3D)

pvo: Potential vorticity (hydrostatic form), PVU. (3D)

qci: Cloud ice mixing ratio, g kg^{-1}. (3D)

> Note: If non-mixed-phase microphysics was used, this field contains the model output cloud water only where $T < 0$ C, and is zero where $T > 0$ C.

qcl: Total cloud mixing ratio, g kg^{-1}. (3D)

> Note: This is just *qcw* + *qci*.

qcw: Cloud water mixing ratio, g kg^{-1}. (3D)

> Note: If non-mixed-phase microphysics was used, this field contains the model output cloud water where $T > 0$ C, and is zero where $T < 0$ C.

qgomfNNN: Full omega (subjected to same processing as QG omega), μ bar s^{-1}. (3D)

> This field is, in principle, the same as *feld=omg*, except it subjects the full omega (i.e., *dp/dt*) field to the same processing as the input fields to QG omega, thus allowing for a meaningful comparison between full omega and QG omega. The actions that *qgomf* are subjected to are an interpolation to a pressure-level grid, a smoothing on that grid (as determined by the value of the keyword *qgsm*---see the explanation for *qgomg* below and the explanation for *qgsm* in Appendix A), and an interpolation back to the model vertical levels.

qgomgNNN or *qmomgNNNRR*: Quasi-geostrophic omega, μ bar s^{-1}. (3D)

> This field is calculated by means of a 3-D inversion (using over-relaxation) of the *Q*-vector form of the quasigeostrophic omega equation, using the domain average Coriolis parameter and vertical stability profile. It is performed on a pressure-level grid, with an irregular lower boundary that matches (as closely as possible) the model topography. The lower boundary condition includes both a topographic BC and an Ekman BC. The three *N*'s correspond to the *Q*-vector forcing, the topographic boundary condition, and the Ekman boundary condition, respectively. Each of the N's should be replaced with either a 1 or 0, to indicate that the corresponding forcing is included or excluded in the omega equation inversion. For example, *feld=qgomg101* would request QG omega calculated from the *Q*-

vector forcing, including the Ekman boundary condition but not the topographic boundary condition. The difference between '*qg*' and '*qm*' is that in the former case, the dry static stability is used everywhere, whereas in the latter case, the moist static stability is used where the relative humidity exceeds a threshold value. The threshold value is specified by replacing '*RR*' with a two digit percent value ranging from 00 to 99.

Note: The first step that RIP takes in calculating this field is the interpolation of geopotential height and specific volume to pressure levels. Following that step, but prior to calculating the *Q*-vector, it is likely that you'll want to have those fields smoothed, so that they represent synoptic or meso-alpha scales. This cannot be accomplished by using *smth* or *smcp,* because the smoothing must occur prior to the *Q*-vector calculation. A special capability has been set up to do this intermediate smoothing. It is accomplished by setting the value of the keyword *qgsm* to a value that indicates the desired number of smoothing passes, as described for the keyword *smth*. So, for example, if you want the intermediate pressure-level fields to be smoothed with 4 passes of the smoother-desmoother, then use *qgsm*=104.

qgr: Graupel mixing ratio (if available as a separate RIP data file), g kg^{-1}. (3D)

qpr: Total precipitation hydrometeor mixing ratio, g kg^{-1}. (3D)

Note: This is just *qra* + *qsn* (+ *qgr* if available).

qra: Rain water mixing ratio, g kg^{-1}. (3D)

Note: If non-mixed-phase microphysics was used, this field contains the model output rain water only where $T > 0$ C., and is zero where $T < 0$ C.

qsn: Snow mixing ratio, g kg^{-1}. (3D)

Note: If non-mixed-phase microphysics was used, this field contains the model output rain water only where $T < 0$ C, and is zero where $T > 0$ C.

qvdiv: Divergence of the Q-vector, 10^{-12} s^{-3} mb^{-1}. (3D)

Note: See the note on intermediate smoothing under the description of the field *qgomg*, above.

qvp: Water vapor mixing ratio, g kg^{-1}. (3D)

qvx: *x*-component of the *Q*-vector, 10^{-6} m s^{-3} mb^{-1}. (3D)

Note: See the note on intermediate smoothing under the description of the field *qgomg*, above.

qvy: *y*-component of the *Q*-vector, 10^{-6} m s^{-3} mb^{-1}. (3D)

Note: See the note on intermediate smoothing under the description of the field *qgomg*, above.

rSSST: Accumulated rainfall (or more generally, precipitation), mm. (2D)

SSS determines the source of the precipitation you're interested in: *cum* for cumulus parameterization precipitation, *exp* for explicitly resolved precipitation,

and *tot* for the sum of cumulus and explicit. *T* (which will actually be two or more characters in length) determines the desired accumulation time period. If it is of the form *Nh*, where *N* is a time period in hours specified either as an integer or as a real number, then RIP will interpret this as "in the past *N* hours". If it is of the form *shN*, then RIP will interpret this as "since hour *N*". An important consideration is that the time that is implied by the specification of *T* must be available in your RIP data set, or else RIP will not be able to access the precipitation file(s) at that time to perform the subtraction. If the time implied by *T* is negative (for example, if you're trying to plot 6-h precipitation accumulation at forecast hour 3), RIP will use the hour-0 data for the subtraction. Here are some examples:

`feld=rcum3h` -> gives cumulus precip in past 3 h
`feld=rtotsh0` -> gives total (cumulus + explicit) precip since hour 0
`feld=rexpsh12.5` -> gives explicit precip since hour 12.5
`feld=rtot24h` -> gives total (cumulus + explicit) in past 24 h; but if plotted before hour 24, gives total precip since hour 0

rhi: Relative humidity with respect to ice, percent. (3D)

rho: Density, kg m^{-3}. (3D)

rhu: Relative humidity with respect to liquid water, percent. (3D)

rib: Near-surface Richardson number, as used in high-resolution PBL scheme, s^{-2}. (2D)

ribbon: Trajectory ribbons (see Chapter 6 for more details).

richnPSSS: Richardson number, dimensionless number. (3D)

> *P* determines the inclusion of moist physics in the calculation of the B-V frequency: *d* for dry, *l* for liquid-only latent heating/cooling considerations, and *i* for liquid and ice latent heating/cooling considerations. *SSS* determines the number of horizontal smoothing passes (on model vertical levels) for the velocity prior to calculation of Rich. num. The *SSS* value can be omitted if no smoothing is desired, i.e. *feld=richnd000* is the same as *feld=richnd*. For an explanation of the smoothing routine, see the description of the keyword *smth* in Appendix A. The component of the wind used for the shear calculation is the along-cross-section (left-to-right) component, where the cross section endpoints are determined by the *crsa* and *crsb* keywords.

sateth: Saturation equivalent potential temperature, K. (3D)

> This is the value of equivalent potential temperature a parcel would have if it were brought to saturation without change in temperature or pressure.

sdpU: Surface dewpoint temperature, units determined by *U*. (3D)

> This field is intended to represent the surface dewpoint temperature as measured by a surface meteorological station (at ~10 m height AGL). From terrain-following data sets, it is calculated with surface pressure, and water vapor mixing ratio at the lowest model level. From pressure- or height-level output, it is

calculated from surface pressure, and from a surface water vapor analysis that RIP looks for in a file ending in *tmk_sfan*. Units can be either in Celsius, Kelvin, or Fahrenheit, depending on whether *U* is set to *c*, *k*, or *f*.

seFFFNNNNN or *smFFFNNNNN*: Various fields derived from inversion of the Sawyer-Eliassen equation, units are variable. (3D)

FFF is either *psi* (Sawyer-Eliassen streamfunction, mb m s^{-1}), *vab* (balanced ageostrophic wind in the plane of the cross section, m s^{-1}), *vtb* (geostrophic plus balanced ageostrophic wind in the plane of the cross section, m s^{-1}), or *omb* (balanced vertical velocity, μ bar s^{-1}). The five *N*'s correspond to five different forcings that can be included in the inversion of the Sawyer-Eliassen equation. They are the confluent frontogenesis term on the RHS, the shear frontogenesis term on the RHS, a topographic boundary condition, an Ekman boundary condition, and a side boundary condition that approximately relates the vertical derivative of ψ to the actual ageostrophic wind along the sides of the cross section. Each of the *N*s should be set to 1 or 0 to indicate that the corresponding forcing is included or excluded in the Sawyer-Eliassen inversion. The difference between '*se*' and '*sm*' is that in the former case, the dry static stability is used everywhere, whereas in the latter case, the moist static stability is used where vertical velocity is upward and the relative humidity exceeds a threshold value. The moist version is solved "double-iteratively", that is to say, the relaxation is carried out to convergence several times, with each of the "big" iterations using the vertical velocity obtained from the previous "big" iteration in order to choose between moist and dry stability at each grid point in the cross section.

A variety of parameters can be adjusted in running the Sawyer-Eliassen diagnosis. These are specified through the keyword *sepa*, which is described in detail in Appendix A.

Note: The Sawyer-Eliassen fields can only be plotted in a vertical cross section with pressure as the vertical coordinate.

sfp: Surface pressure, mb. (2D)

slp: Sea-level pressure inspired by Benjamin and Miller (1990), mb. (2D)

snf: Frozen fraction of precipitation hydrometeor mixing ratios, dimensionless number between 0.0 and 1.0. (3D)

Note: This is just [*qsn* (+ *qgr* if available)] / *qpr*. If *qpr* is less than .0001 g kg^{-1}, then *snf* is set to 0.5. This can be used to give a nice depiction of the "rain/snow line" if mixed phase microphysics are used.

sno: Snow cover (if available as a separate RIP data file). (2D)

spsqTPSSS: Scorer parameter squared, m^{-2}. (3D)

T determines which terms to include: *a* for $N^2/(u-c)^2$, *b* for $u_{zz}/(u-c)$, or *t* for total (*a* + *b*). *P* determines the inclusion of moist physics in the calculation of the B-V frequency: *d* for dry, *l* for liquid-only latent heating/cooling considerations, and *i* for liquid and ice latent heating/cooling considerations. *SSS* determines the

number of horizontal smoothing passes (on model vertical levels) for the velocity prior to calculation of Scorer parameter. The *SSS* value can be omitted if no smoothing is desired, i.e. *feld=spsqtd000* is the same as *feld=spsqtd*. For an explanation of the smoothing routine, see the description of the *smth* keyword in Appendix A. The component of the wind used for the shear calculation is the along-cross-section (left-to-right) component, where the cross section endpoints are determined by the *crsa* and *crsb* keywords. The phase speed in terms *a* and *b* can be provided with the *strm* keyword. See the description of the *strm* keyword in Appendix A.

sr9: Storm-relative flow at 9 km AGL, m s^{-1}. (2D)

sreh: Storm-relative environmental helicity, m^2 s^{-2}. (2D)

> Assumes a supercell storm moves at 75% of the magnitude of, and 30 to the right of, the mean wind vector between 3 and 10 km AGL.

srfh: Storm-relative flow at high levels (~9 km), m^2 s^{-2}. (2D)

> Assumes a supercell storm moves at 75% of the magnitude of, and 30 to the right of, the mean wind vector between 3 and 10 km AGL.

srfl: Storm-relative flow at low levels (~2 km), m^2 s^{-2}. (2D)

> Assumes a supercell storm moves at 75% of the magnitude of, and 30 to the right of, the mean wind vector between 3 and 10 km AGL.

stb: Static stability [$-d\theta/dp$], K hPa^{-1}. (3D)

stbe: Potential or convective stability [$-d\theta_e/dp$], K hPa^{-1}. (3D)

stbz: Static stability [$d\theta/dz$], K km^{-1}. (3D)

swarm: Trajectory swarm (see Chapter 6 for more details).

tdd: Dew point depression, C. (3D)

tdf: Dew point temperature, F. (3D)

tdk: Dew point temperature, K. (3D)

tdp: Dew point temperature, C. (3D)

tdsfU: Surface dew point temperature, units are variable depending on the value of *U*. (2D)

> It is computed using 2 m water vapor mixing ratio if available. Otherwise it uses the moisture field from the lowest model level. Units of *tdsfU* can be either in Celsius, Kelvin, or Fahrenheit, depending on whether *U* is set to *c*, *k*, or *f*.

ter: Terrain height, m. (2D)

tfp: Frost point temperature, C. (3D)

tgc: Ground temperature, C. (2D)

tgk: Ground temperature, K. (2D)

thckMMMNNN: Thickness, decameters (or dam). (2D)

> *MMM* and *NNN* are the lower and upper pressure levels, respectively, expressed in kPa with leading zeros included, so that the total number of digits is always 6. For example:
>
> ```
> thck100050  -> This gives the 1000 to 500 hPa thickness.
> thck050005  -> This gives the 500 to 50 hPa thickness.
> ```

the: Potential temperature, K. (3D)

thsf: Surface air potential temperature, K. (2D)

> This is intended to represent the surface air potential temperature as measured by a surface meteorological station (at ~10 m height AGL). The surface air temperature is first calculated, as described for field *tsfU*. It is then converted to potential temperature.

thv: Virtual potential temperature, K. (3D)

thvhm: Virtual potential temperature, with hydrometeor effect included, K. (3D)

> This is the same as *thv*, except it is virtual potential temperature including the negative effects of hydrometeors.

tmc: Temperature, C. (3D)

tmclKK: Temperature of a lifted parcel, C. (3D)

> The level from which parcels are to be lifted is the model level that is the *KK*th level from the bottom (always use two digits). Therefore, this field is only defined for model levels at or above that level. This field is most useful for plotting in a sounding, to give a sense of the CAPE.

tmf: Temperature, F. (3D)

tmk: Temperature, K. (3D)

tpt: Temperature perturbation, C (or K). (3D)

> The perturbation is with respect to the reference state, as defined by the keyword *rfst*.

tptslr: Temperature perturbation with respect to the U.S. Standard Atmosphere, C. (or K) (3D)

track: Typhoon track

> Objectively-determined typhoon track including central pressures. The default code is able to track five storms simultaneously. The track is plotted using all times in the .xtimes file, regardless of the value of ptimes. The typhoon symbol and central pressure locations are plotted based on the keyword tynt. The typhoon location, central pressure, and maximum surface wind speed is printed out at each forecast output time. While the track algorithm is designed for use with any model grid spacing, performance for any particular delta-x may be improved by tuning the

algorithm. Additional documentation is included in the source code (including a sample plspec entry).

tsfU: Surface air temperature, units are variable depending on the value of *U*. (2D)

> If 2 m temperature is available from the dataset, this is that temperature. Otherwise, this temperature is intended to represent the surface air temperature as measured by a surface meteorological station (at ~10 m height AGL). From terrain-following data sets, it is calculated with a crude application of similarity theory: if the air temperature at the lowest model layer (T_{LML}) is warmer than the ground temperature, *tsfc* is equal to T_{LML}; if T_{LML} is colder than the ground temperature, *tsfc* is equal to the average of the ground temperature and T_{LML}. From pressure- or height-level output, it looks for the surface air temperature analysis that is stored in the file ending in *tmk_sfan*. Units of *tsfU* can be either in Celsius, Kelvin, or Fahrenheit, depending on whether *U* is set to *c*, *k*, or *f*.

tvc: Virtual temperature, C. (3D)

tvclKK: Virtual temperature of a lifted parcel, C. (3D)

> This is the same as *tmcl*, except it gives the virtual temperature of a lifted parcel. When plotted alongside *tvc* on a skew-T sounding, this gives a more accurate graphical representation of CAPE than does *tmcl* plotted alongside *tmc*.

tvk: Virtual temperature, K. (3D)

tvp: Virtual temperature perturbation, C (or K). (3D)

> This is the same as *tpt*, except it is the perturbation of virtual temperature with respect to the reference state, as defined by the keyword *rfst*.

tvphm: Virtual temperature perturbation, with hydrometeor effect included, C (or K). (3D)

> This is the same as *tvp*, except it is the perturbation of virtual temperature including the negative effects of hydrometeors on virtual temperature.

twb: Wet bulb temperature, C. (3D)

uageo: Horizontal ageostrophic wind speed in the *x* (or *j*-index) direction, m s^{-1}. (3D)

ugeo: Horizontal geostrophic wind speed in the *x* (or *j*-index) direction, m s^{-1}. (3D)

unor: Westerly horizontal wind component, m s^{-1}. (3D)

uubs: 0-6 km shear (*x*-component), m s^{-1}. (2D)

uusr: Supercell motion vector following Bukers et al (1999) (*x*-component), m s^{-1}. (2D)

uuu: Horizontal wind speed in the *x* (or *j*-index) direction, m s^{-1}. (3D)

vadvtheta: Vertical advection of potential temperature, K h^{-1}. (3D)

> This calculates the local change in potential temperature due to vertical advection. The vertical velocity and potential temperature fields can be smoothed on

constant-pressure surfaces prior to the computing of the product in this nonlinear quantity, by using the *qgsm* keyword.

vageo: Horizontal ageostrophic wind speed in the *y* (or *i*-index) direction, m s^{-1}. (3D)

vgeo: Horizontal geostrophic wind speed in the *y* (or *i*-index) direction, m s^{-1}. (3D)

vgp: Vorticity generation potential, m s^{-2}. (2D)

> This is the 0 to 3 km AGL total shear (i.e., the length of the 0-3 km AGL hodograph divided by 3000 m) times the square root of CAPE of the parcel with maximum θ_e below 3 km AGL.

visX: Surface visibility due to hydrometeors, km. (2D)

> Regardless of whether mixed phase microphysics were used, there are three possibilities to choose from for the determination of the phase of cloud and precipitation hydrometeors: Replace *X* with *d* for the 0 Celsius assumption, *b* for the Bocchieri assumption, or *r* for the mixed phase assumption. See code for details of these assumptions.

vnor: Southerly horizontal wind component, m s^{-1}. (3D)

vor: Relative vorticity (vertical component), 10^{-5} s^{-1}. (3D)

vox: *x*-component of vorticity, 10^{-5} s^{-1}. (3D)

voy: *y*-component of vorticity, 10^{-5} s^{-1}. (3D)

vvbs: 0-6 km shear (*y*-component), m s^{-1}. (2D)

vvsr: Supercell motion vector following Bukers et al (1999) (*y*-component), m s^{-1}. (2D)

vvv: Horizontal wind speed in the *y* (or *i*-index) direction, m s^{-1}. (3D)

wdr: Horizontal wind direction (compass direction that wind is blowing from), degrees. (3D)

wsp: Horizontal wind speed, m s^{-1}. (3D)

www: Vertical velocity (*dz/dt*), cm s^{-1}. (3D)

xlat: Latitude (if available as a separate RIP data file). (2D)

xlon: Longitude (if available as a separate RIP data file). (2D)

xlus: Land use category (if available as a separate RIP data file). (2D)

xmap: Map factor, dimensionless number. (2D)

xnt: Horizontal wind component normal to the cross section, m s^{-1}. (3D)

> Note: The normal wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be into the cross section.

xntageo: Horizontal ageostrophic wind component normal to the cross section, m s^{-1}. (3D)

Note: The normal wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be into the cross section.

xntgeo: Horizontal geostrophic wind component normal to the cross section, m s^{-1}. (3D)

Note: The normal wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be into the cross section.

xpt: Horizontal wind component parallel to the cross section, m s^{-1}. (3D)

Note: The parallel wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be from left to right.

xptageo: Horizontal ageostrophic wind component parallel to the cross section, m s^{-1}. (3D)

Note: The parallel wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be from left to right.

xptgeo: Horizontal geostrophic wind component parallel to the cross section, m s^{-1}. (3D)

Note: The parallel wind component is calculated using the cross section end points defined by the *crsa* and *crsb* keywords. Positive is assumed to be from left to right.

Appendix C. Format of RIP data files

When RIPDP is executed, it produces a large number of data files. This appendix describes the naming convention and format of RIP data files.

a. The .xtimes file

A text file is created, with the name `model-data-set-name.xtimes`, where `model-data-set-name` is as defined in Section 3. On the first line of the file is the total number of times available in the RIP data set. Beneath the first line is a list of those times (one time per line). Each time is expressed as a forecast hour with four digits left of the decimal and five to the right.

b. The .minfo file

A text file is created, with the name `model-data-set-name.minfo`. This file contains one or more lines of textual information about the model run, which can be read in by RIP and plotted at the bottom of each plot produced by RIP, to give the viewer some basic information about the model run.

c. The data files

The data files are named with the following convention:

`model-data-set-name` _ `HHHH.HHHHH` _ `variable`

`model-data-set-name` is as defined in [Section 3]. `HHHH.HHHHH` is the forecast time in hours (four digits left of the decimal and five to the right). `variable` is the name of the variable that is contained in the file. These three pieces of the file name are separated by underscore characters.

The format of the file is perhaps best described by showing the Fortran 77 code with which it can be written:

```
c
c    Declare RIP header variables
c
      integer ihrip(32)
      real rhrip(32)
      character chrip(64)*64,vardesc*64,plchun*24
c
c    Declare 3D and 2D (horizontal slab) data arrays
c
      real data3(maxiy,maxjx,maxkz), data2(maxiy,maxjx)
c
c    Open the file, write header and data, close the file
c
      open
(unit=10,file=<filename>,form='unformatted',status='new')
      write (10) vardesc,plchun,ihrip,rhrip,chrip
      write (10) data3
c  or write (10) data2    for a 2D field
      close (10)
```

`maxiy` and `maxjx` are the dimensions of the dot-point gird, regardless of whether the variable is defined on dot points or cross points. `maxkz` is the number of vertical levels. All 3D variables in a RIP data set must be defined on the same vertical levels. For example, in the MM5 vertical coordinate system, most of the variables are defined on the staggered "half-sigma" levels, but vertical velocity is defined on the "full-sigma" levels. To adhere to the "same vertical levels" requirement, *ripdp_mm5* interpolates the vertical velocities to the "half-sigma" levels. `maxkz` would then be equal to the number of half-sigma levels.

`vardesc` is a character string, 64 characters maximum length, that contains a brief verbal description of the variable. It may also include units, although the representation of units is limited to the ASCII character set.

`plchun` is a character string, 24 characters maximum length, that contains the units of the variable, in NCAR Graphics PLOTCHAR code. This code can be used by the NCAR Graphics code in RIP to display the units of the variable on a plot, including special characters (e.g., Greek symbols) and exponents.

`ihrip` and `rhrip` are integer and real header arrays (respectively) contain information about model dimensions, start time and date, and map background, as well as information about the variable such as a brief description and units. The `chrip` array contains a text description, 64 characters maximum length, of each of the elements of the `ihrip` and `rhrip` arrays. Elements 1-32 of `chrip` correspond to the 32 elements of `ihrip`, and elements 33-64 of `chrip` correspond to the 32 elements or `rhrip`. Note that not all of the elements of `ihrip` and `rhrip` are used. A list of all the currently defined elements of `ihrip` and `rhrip`, as well as the text description in the corresponding element of `chrip`, are shown in the table below:

Header Array Element	Description
ihrip(1)	chrip(1)='map projection (0: none/ideal, 1: LC, 2: PS, 3: ME, 4: SRCE)'
ihrip(2)	chrip(2)='number of dot points in the y-direction (coarse domain)'
ihrip(3)	chrip(3)='number of dot points in the x-direction (coarse domain) '
ihrip(4)	chrip(4)='number of dot points in the y-direction (this domain) '
ihrip(5)	chrip(5)='number of dot points in the x-direction (this domain) '
ihrip(6)	chrip(6)='number of dimensions of this variable (2 or 3) '
ihrip(7)	chrip(7)='grid of this variable (1: cross point dom., 0: dot point dom.) '
ihrip(9)	chrip(9)='number of vertical levels in the data'
ihrip(10)	chrip(10)='mdateb: YYMMDDHH (truncated hour) of hour-0 for this dataset'
ihrip(11)	chrip(11)='mdate: YYMMDDHH (truncated hour) of this time'
ihrip(12)	chrip(12)='ice physics (1: sep. arrays for ice fields, 0: no sep. arrays) '
ihrip(13)	chrip(13)='ver. coord. type: <or=3: hgt. or prs.; >or=4: terrain-following '
ihrip(14)	chrip(14)='landuse dataset (1: old, 13-cat; 2: USGS, 24-cat; 3: SiB, 16) '
rhrip(1)	chrip(33)='first true latitude (deg.) '
rhrip(2)	chrip(34)='second true latitude (deg.) '
rhrip(3)	chrip(35)='central latitude of coarse domain (deg.) '
rhrip(4)	chrip(36)='central longitude of coarse domain(deg.) '
rhrip(5)	chrip(37)='grid distance of coarse domain (km) '
rhrip(6)	chrip(38)='grid distance of this domain (km) '
rhrip(7)	chrip(39)='coarse dom. y-position of lower left corner of this domain '
rhrip(8)	chrip(40)='coarse dom. x-position of lower left corner of

	this domain '
rhrip(13)	chrip(45)='rhourb: diff (in h) between exact time and mdate of hour-0 '
rhrip(14)	chrip(46)='rhour: diff (in h) between exact time and mdate of this data '
rhrip(15)	chrip(47)='xtime: exact time of this data relative to exact hour-0 (in h) '

If you want to create a version of RIPDP that converts output from a given model into a RIP data set, there are certain rules that apply to the variable that RIPDP can or should create, and the variables expected by RIP. The data variables can be classified into three different categories:

Required variables

Optional variables

Unexpected variables

Required variables are fundamental variables that RIP expects to find at every time period that will be processed. If it does not, it will abort. Therefore, RIPDP should create these variables exactly as shown in the table below, particularly in terms of variable name, dimensionality, and grid (cross or dot-point).

Table of Required Variables

Variable	Dimensions [ihrip(6)]	Grid [ihrip(7)]	Description of variable (vardesc)	Units of variable in PLOTCHAR code (plchun)
uuu	3	0	Horizontal wind (y-comp.), m/s	m s~S~-1~N~
vvv	3	0	Horizontal wind (y-comp.), m/s	m s~S~-1~N~
tmk	3	1	Temperature, K	K
ght	3	1	Geopotential height, m	m
prs	3	1	Pressure, hPa	hPa
ter	2	1	Terrain height AMSL, m	m
sfp	2	1	Surface pressure, hPa	hPa
xmap	2	1	Map factor on cross points	none
dmap	2	0	Map factor on dot points	none
cor	2	1	Coriolis parameter, per s	s~S~-1~N~

Optional variables are variables that RIP might expect to find depending on the field(s) requested in the user input file. However, they are not required for RIP to run. If any of these variables is requested directly, or if a field is requested that requires one of the optional variables, in some cases the variable will be assumed to be zero, and in other cases RIP will abort. If you are creating a version of RIPDP for your model, it is a good idea to have it create these optional variables if they are available, and to make sure they agree with the attributes shown in the table below. This will improve the functionality of RIP. For example, there is a field in RIP called *qpr*, which is the sum of all the precipitation mixing ratio fields. When this field is calculated, RIP specifically looks for files with variable names *qra*, *qsn*, and *qgr*. If they do not exist, or if they exist but are named something else, then the *qpr* field will not work.

Table of Optional Variables

Variable	Dimensions [ihrip(6)]	Grid [ihrip(7)]	Description of variable (vardesc)	Units of variable in PLOTCHAR code (plchun)
qvp	3	1	Water vapor mixing ratio, g/kg	g kg~S~-1~N~
www	3	1	Vertical velocity, cm/s	cm s~S~-1~N~
qcw	3	1	Cloud water mixing ratio, g/kg	g kg~S~-1~N~
qra	3	1	Rain water mixing ratio, g/kg	g kg~S~-1~N~
qci	3	1	Cloud ice mixing ratio, g/kg	g kg~S~-1~N~
qsn	3	1	Snow mixing ratio, g/kg	g kg~S~-1~N~
qgr	3	1	Graupel mixing ratio, g/kg	g kg~S~-1~N~
rtc	2	1	Cumulus precip. since h 0, mm	Mm
rte	2	1	Explicit precip. since h 0, mm	Mm
tgk	2	1	Ground/sea-surface temperature, K	K
tmk_sfan	2	1	Temperature (sfc. anal.), K	K
qvp_sfan	2	1	Water vapor mixing ratio (sfc. anal.), g/kg	g kg~S~-1~N~
xlat	2	1	Latitude, degrees	degrees
xlon	2	1	Longitude, degrees	degrees
xlus	2	1	Land use category	none

Finally, an unexpected variable is a variable that RIPDP does not specifically expect to find in the model output, but upon encountering it, processes it and creates a data file. The variable name, dimensionality, grid, description, and units are determined from

metadata in the model output file. These variables can be plotted by RIP by using the automatically determined variable name in the *feld=* plot specifier.