

User's Guide for the NMM Core of the Weather Research and Forecast (WRF) Modeling System Version 3

Chapter 7: Post Processing Utilities

Table of Contents

[NCEP WRF Post Processor \(WPP\)](#)

- [WPP Introduction](#)
- [WPP Required Software](#)
- [Obtaining the WPP Code](#)
- [WPP Directory Structure](#)
- [Installing the WPP Code](#)
- [WPP Functionalities](#)
- [Computational Aspects and Supported Platforms for WPP](#)
- [Setting up the WRF model to interface with WPP](#)
- [WPP Control File Overview](#)
 - [Controlling which variables *wrfpost* outputs](#)
 - [Controlling which levels *wrfpost* outputs](#)
- [Running WPP](#)
 - [Overview of the steps in scripts to run WPP](#)
- [Visualization with WPP](#)
 - [GEMPAK](#)
 - [GrADS](#)
- [Fields Produced by *wrfpost*](#)

[RIP4](#)

- [RIP Introduction](#)
- [RIP Software Requirements](#)
- [RIP Environment Settings](#)
- [Obtaining the RIP Code](#)
- [RIP Directory Structure](#)
- [Installing the RIP Code](#)
- [RIP Functionalities](#)
- [RIP Data Preparation \(RIPDP\)](#)
 - [RIPDP Namelist](#)
 - [Running RIPDP](#)
- [RIP User Input File \(UIF\)](#)
- [Running RIP](#)
 - [Calculating and Plotting Trajectories with RIP](#)
 - [Creating Vis5D Datasets with RIP](#)

NCEP WRF Postprocessor (WPP)

WPP Introduction

The NCEP WRF Postprocessor was designed to interpolate both WRF-NMM and WRF-ARW output from their native grids to National Weather Service (NWS) standard levels (pressure, height, etc.) and standard output grids (AWIPS, Lambert Conformal, polar-stereographic, etc.) in NWS and World Meteorological Organization (WMO) GRIB format. This package also provides an option to output fields on the model's native vertical levels.

The adaptation of the original WRF Postprocessor package and User's Guide (by Mike Baldwin of NSSL/CIMMS and Hui-Ya Chuang of NCEP/EMC) was done by Lígia Bernardet (NOAA/ESRL/DTC) in collaboration with Dusan Jovic (NCEP/EMC), Robert Rozumalski (COMET), Wesley Ebisuzaki (NWS/HQTR), and Louisa Nance (NCAR/DTC). Upgrades to WRF Postprocessor versions 2.2 and higher were performed by Hui-Ya Chuang and Dusan Jovic (NCEP/EMC).

WPP Software Requirements

The WRF Postprocessor requires the same Fortran and C compilers used to build the WRF model. In addition to the netCDF library, the WRF I/O API libraries, which are included in the WRF model tar file, are also required.

The WRF Postprocessor has some visualization scripts included to create graphics using either GrADS (<http://grads.iges.org/grads/grads.html>) or GEMPAK (<http://www.unidata.ucar.edu/software/gempak/index.html>). These packages are not part of the WPP installation and would need to be installed.

Obtaining the WPP Code

The WRF Postprocessor package can be downloaded from: <http://www.dtcenter.org/wrf-nmm/users/downloads/>.

Note: Always obtain the latest version of the code if you are not trying to continue a pre-existing project. WPPV3 is just used as an example here.

Once the *tar* file is obtained, *gunzip* and *untar* the file.

```
tar -zxvf WPPV3.tar.gz
```

This command will create a directory called **WPPV3**.

WPP Directory Structure

Under the main directory of *WPPV3* reside five subdirectories:

src: contains source codes for *wrfpost*, *ndate*, and *copygb*.

scripts: contains sample running scripts

run_wrfpost: run *wrfpost* and *copygb*.

run_wrfpostandgempak: run *wrfpost*, *copygb*, and GEMPAK to plot various fields.

run_wrfpostandgrads: run *wrfpost*, *copygb*, and GrADS to plot various fields.

run_wrfpost_frames: run *wrfpost* and *copygb* on a single *wrfout* file containing multiple forecast times.

run_wrfpost_gracet: run *wrfpost* and *copygb* on *wrfout* files with non-zero minutes/seconds.

run_wrfpost_minute: run *wrfpost* and *copygb* for sub-hourly *wrfout* files.

lib: contains source code subdirectories for the WRF Postprocessor libraries and is the directory where the WRF Postprocessor compiled libraries will reside.

w3lib: Library for coding and decoding data in GRIB format

Note: The version of this library included in this package is Endian-independent and can be used on LINUX and IBM systems.

iplib: General interpolation library (see *lib/iplib/iplib.doc*)

splib: Spectral transform library (see *lib/splib/splib.doc*)

wrfmpi_stubs: Contains some *C* and *FORTRAN* codes to generate *libmpi.a* library. It supports MPI implementation for LINUX applications.

parm: Contains the parameter files, which can be modified by the user to control how the post processing is performed.

exec: Location of executables after compilation.

Installing the WPP Code

WPP uses a build mechanism similar to that used by the WRF model. First issue the *configure* command, followed by the *compile* command.

If the WRFV3 directory is not located at:

```
../WRFV3
```

the following environment variable must be set:

```
setenv WRF_DIR /home/user/WRFV3
```

If this is not set, the configure script will prompt you for it.

Type *configure*, and provide the required info. For example:

./configure

You will be given a list of choices for your computer.

Choices for IBM machines are as follows:

1. AIX xlf compiler with xlc (serial)

Choices for LINUX operating systems are as follows:

1. LINUX i486 i586 i686, PGI compiler (serial)
2. LINUX i486 i586 i686, Intel compiler (serial)
3. LINUX i486 i586 i686, gfortran compiler (serial)

Choose one of the configure options listed. Check the *configure.wpp* file created and edit for compile options/paths, if necessary.

To compile WPP, enter the following command:

```
./compile >& compile_wpp.log &
```

This command should create four WRF Postprocessor libraries in *lib/* (*libmpi.a*, *libsp.a*, *libip.a*, and *libw3.a*) and three WRF Postprocessor executables in *exec/* (*wrfpost.exe*, *ndate.exe*, and *copygb.exe*).

To remove all built files, as well as the *configure.wpp*, type:

```
clean
```

This action is recommended if a mistake is made during the installation process.

WPP Functionalities

The WRF Postprocessor V3,

- is compatible with WRF v2.2 and higher.
- can be used to post-process both WRF-ARW and WRF-NMM forecasts.
- can ingest WRF history files (*wrfout**) in two formats: netCDF and binary.

The WRF Postprocessor is divided into two parts:

1. *Wrfpost*

- Interpolates the forecasts from the model's native vertical coordinate to NWS standard output levels (e.g., pressure, height) and computes mean sea level pressure. If the requested parameter is on a model's native level, then no vertical interpolation is performed.
- Computes diagnostic output quantities (e.g., convective available potential energy, helicity, radar reflectivity). A list of fields that can be generated by *wrfpost* is shown in Table 2.

- Outputs the results in NWS and WMO standard GRIB1 format (for GRIB documentation, see <http://www.nco.ncep.noaa.gov/pmb/docs/>).
- Destaggers the WRF-ARW forecasts from a C-grid to an A-grid.
- Outputs two navigation files, *copygb_nav.txt* and *copygb_hwrf.txt* for WRF-NMM. These files can be used as input for *copygb*.
 - *copygb_nav.txt*: This file contains the GRID GDS of a Lambert Conformal Grid similar in domain and grid spacing to the one used to run the WRF model. The Lambert Conformal map projection works well for mid-latitudes.
 - *copygb_hwrf.txt*: This file contains the GRID GDS of a Latitude-Longitude Grid similar in domain and grid spacing to the one used to run the WRF model. The latitude-longitude grid works well for tropics.

2. Copygb

- Destaggers the WRF-NMM forecasts from the staggered native E-grid to a regular non-staggered grid.
(Since *wrfpost* destaggers WRF-ARW output from a C-grid to an A-grid, WRF-ARW data can be displayed directly without going through *copygb*.)
- Interpolates the forecasts horizontally from their native grid to a standard AWIPS or user-defined grid (for information on AWIPS grids, see <http://www.nco.ncep.noaa.gov/pmb/docs/on388/tableb.html>).
- Outputs the results in NWS and WMO standard GRIB1 format (for GRIB documentation, see <http://www.nco.ncep.noaa.gov/pmb/docs/>).

In addition to *wrfpost* and *copygb*, a utility called *ndate* is distributed with the WRF Postprocessor tarfile. This utility is used to format the dates of the forecasts to be posted for ingestion by the codes.

Computational Aspects and Supported Platforms for WPP

The WRF Postprocessor v3.0 has been tested on IBM and LINUX platforms. Only *wrfpost* (step 1) is parallelized because it requires several 3-dimensional arrays (the model's history variables) for the computations. When running *wrfpost* on more than one processor, the last processor will be designated as an I/O node, while the rest of the processors are designated as computational nodes. For example, if three processors are requested to run the *wrfpost*, only the first two processors will be used for computation, while the third processor will be used to write output to GRIB files.

Setting up the WRF model to interface with WPP

The *wrfpost* program is currently set up to read a large number of fields from the WRF model history files. This configuration stems from NCEP's need to generate all of its required operational products. A list of the fields that are currently read in by *wrfpost* is provided for the WRF-NMM in Table 1 and for the WRF-ARW in Table 2. This program is configured such that it will run successfully if an expected input field is missing from the WRF history file as long as this field is not required to produce a

requested output field. If the pre-requisites for a requested output field are missing from the WRF history file, *wrfpost* will abort at run time.

Take care not to remove fields from the *wrfout* files, which may be needed for diagnostic purposes by the WPP package. For example, if isobaric state fields are requested, but the pressure fields on model interfaces (PINT for WRF-NMM, P and PB for WRF-ARW) are not available in the history file, *wrfpost* will abort at run time. In general, the default fields available in the *wrfout* files are sufficient to run WPP. The fields written to the WRF history file are controlled by the settings in the Registry file (see *Registry.EM* or *Registry.NMM(_NEST)* files in the *Registry* subdirectory of the main *WRFV3* directory).

Note: It is necessary to re-compile the WRF model source code after modifying the appropriate Registry file.

Table 1. List of all possible fields read in by *wrfpost* for the WRF-NMM:

T	SFCEXC	NRDSW
U	VEGFRC	ARDSW
V	ACSNOW	ALWIN
Q	ACSNOM	ALWOUT
CWM	CMC	NRDLW
F_ICE	SST	ARDLW
F_RAIN	EXCH_H	ALWTOA
F_RIMEF	EL_MYJ	ASWTOA
W	THZ0	TGROUND
PINT	QZ0	SOILTB
PT	UZ0	TWBS
PDTOP	VZ0	SFCSHX
FIS	QS	NSRFC
SMC	Z0	ASRFC
SH2O	PBLH	QWBS
STC	USTAR	SFCLHX
CFRACH	AKHS_OUT	GRNFLX
CFRACL	AKMS_OUT	SUBSHX
CFRACM	THS	POTEVP
SLDPTH	PREC	WEASD
U10	CUPREC	SNO
V10	ACPREC	SI
TH10	CUPPT	PCTSNO
Q10	LSPA	IVGTYP
TSHLTR	CLDEFI	ISLTYP
QSHLTR	HTOP	ISLOPE
PSHLTR	HBOT	SM

SMSTAV	HTOPD	SICE
SMSTOT	HBOTD	ALBEDO
ACFRCV	HTOPS	ALBASE
ACFRST	HBOTS	GLAT
RLWTT	SR	XLONG
RSWTT	RSWIN	GLON
AVRAIN	CZEN	DX_NMM
AVCNVC	CZMEAN	NPHS0
TCUCN	RSWOUT	NCLOD
TRAIN	RLWIN	NPREC
NCFRCV	SIGT4	NHEAT
NCFRST	RADOT	
SFROFF	ASWIN	
UDROFF	ASWOUT	
SFCEVP		

Note: For WRF-NMM, the period of accumulated precipitation is controlled by the *namelist.input* variable *tprec*. Hence, this field in the *wrfout* file represents an accumulation over the time period $tprec * INT[(fhr - \Sigma) / tprec]$ to fhr, where fhr represents the forecast hour and Σ is a small number. The GRIB file output by *wrfpost* and by *copygb* contains fields with the name of accumulation period.

Table 2. List of all possible fields read in by *wrfpost* for the WRF-ARW:

T	MUB	SFROFF
U	P_TOP	UDROFF
V	PHB	SFCEVP
QVAPOR	PH	SFCEXC
QCLOUD	SMOIS	VEGFRA
QICE	TSLB	ACSNOW
QRAIN	CLDFRA	ACSNOM
QSNOW	U10	CANWAT
QGRAUP	V10	SST
W	TH2	THZ0
PB	Q2	QZ0
P	SMSTAV	UZ0
MU	SMSTOT	VZ0
QSFC	HGT	ISLTYP
Z0	ALBEDO	ISLOPE
UST	GSW	XLAND
AKHS	GLW	XLAT
AKMS	TMN	XLONG
TSK	HFX	MAPFAC_M
RAINC	LH	STEPBL
RAINNC	GRDFLX	HTOP

RAINCV	SNOW	HBOT
RAINNCV	SNOWC	

Note: For WRF-ARW, the accumulated precipitation fields (*RAINCV* and *RAINNCV*) are run total accumulations.

WPP Control File Overview

The user interacts with *wrfpost* through the control file, *parm/wrf_cntrl.parm*. The control file is composed of a header and a body. The header specifies the output file information. The body allows the user to select which fields and levels to process.

The header of the *wrf_cntrl.parm* file contains the following variables:

- **KGTYPE:** defines output grid type, which should always be 255.
- **IMDLTY:** identifies the process ID for AWIPS.
- **DATSET:** defines the prefix used for the output file name. Currently set to “*WRFPRS*”.

The body of the *wrf_cntrl.parm* file is composed of a series of line pairs similar to the following:

```
(PRESS ON MDL SFCS ) SCAL=( 3.0)
L=(11000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000)
```

where,

- the top line specifies the variable (e.g. PRESS) to process, the level type (e.g. ON MDL SFCS) a user is interested in, and the degree of accuracy to be retained (SCAL=3.0) in the GRIB output.
 - SCAL defines the precision of the data written out to the GRIB format. Positive values denote decimal scaling (maintain that number of significant digits), while negative values describe binary scaling (precise to $2^{\{SCAL\}}$; i.e., SCAL=-3.0 gives output precise to the nearest 1/8).

A list of all possible output fields for *wrfpost* is provided in Table 3. This table provides the full name of the variable in the first column and an abbreviated name in the second column. The abbreviated names are used in the control file. Note that the variable names also contain the type of level on which they are output. For instance, temperature is available on “model surface” and “pressure surface”.
- The second line specifies the levels on which the variable is to be posted.

Controlling which variables *wrfpost* outputs

To output a field, the body of the control file needs to contain an entry for the appropriate variable and output for this variable must be turned on for at least one level (see “*Controlling which levels wrfpost outputs*”). If an entry for a particular field is not yet

available in the control file, two lines may be added to the control file with the appropriate entries for that field.

Controlling which levels *wrfpost* outputs

The second line of each pair determines which levels *wrfpost* will output. Output on a given level is turned off by a “0” or turned on by a “1”.

- For isobaric output, 47 levels are possible, from 2 to 1013 hPa (*8 levels above 75 mb and then every 25 mb from 75 to 1000 mb*). The complete list of levels is specified in *src/wrfpost/POSTDATA.f*.
 - Modify specification of variable LSM in the file CTLBLK.com to change the number of pressure levels: PARAMETER (LSM=47)
 - Modify specification of SPL array in the subroutine *POSTDATA.f* to change the values of pressure levels:
DATA SPL/200.,500.,700.,1000.,2000.,3000.
&,5000.,7000.,7500.,10000.,12500.,15000.,17500.,20000., ...
- For model-level output, all model levels are possible, from the highest to the lowest.
- When using the Noah LSM, the *soil layers* are 0-10 cm, 10-40 cm, 40-100 cm, and 100-200 cm.
- When using the RUC LSM, the *soil levels* are 0 cm, 5 cm, 20 cm, 40 cm, 160 cm, and 300 cm. For the RUC LSM it is also necessary to turn on two additional output levels in the *wrf_cntrl.parm* to output 6 levels rather than the default 4 layers for the Noah LSM.
- For PBL layer averages, the levels correspond to 6 layers with a thickness of 30 hPa each.
- For flight level, the levels are 914 m,1524 m,1829 m, 2134 m, 2743 m, 3658 m, and 6000 m.
- For AGL RADAR Reflectivity, the levels are 4000 and 1000 m.
- For surface or shelter-level output, only the first position of the line needs to be turned on.
 - For example, the sample control file *parm/wrf_cntrl.parm* has the following entry for surface dew point temperature:

```
(SURFACE DEWPOINT  ) SCAL=( 4.0)
L=(00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000)
```

Based on this entry, surface dew point temperature will not be output by *wrfpost*. To add this field to the output, modify the entry to read:

```
(SURFACE DEWPOINT  ) SCAL=( 4.0)
L=(10000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000)
```

Running WPP

Four scripts for running the WRF Postprocessor package are included in the tar file:

```
run_wrfpost  
run_wrfpostandgrads  
run_wrfpostandgempak  
run_wrfpost_frames  
run_wrfpost_gracet  
run_wrfpost_minute
```

Before running any of the above listed scripts, perform the following instructions:

1. *cd* to your *DOMAINPATH* directory.
2. Make a directory to put the WRF Postprocessor results.

```
mkdir postprd
```

3. Make a directory to put a copy of *wrf_cntrl.parm* file.

```
mkdir parm
```

4. Copy over the default *WPPV3/parm/wrf_cntrl.parm* to your working directory to customize *wrfpost*.
5. Edit the *wrf_cntrl.parm* file to reflect the fields and levels you want *wrfpost* to output.
6. Copy over the (*WPPV3/scripts/run_wrfpost**) script of your choice to the *postprd/*.
7. Edit the run script as outlined below.

Once these directories are set up and the edits outlined above are completed, the scripts can be run interactively from the *postprd* directory by simply typing the script name on the command line.

Overview of the scripts to run the WPP

Note: It is recommended that the user refer to the script while reading this overview.

1. Set up environmental variables:
TOP_DIR: top level directory for source codes (*WPPV3* and *WRFV3*)
DOMAINPATH: top level directory of WRF model run

Note: The scripts are configured such that *wrfpost* expects the WRF history files (*wrfout** files) to be in subdirectory *wrfprd*, the *wrf_cntrl.parm* file to be in the subdirectory *parm* and the postprocessor working directory to be a subdirectory

called *postprd* under *DOMAINPATH*.

2. Specify dynamic core being run (“NMM” or “ARW”)
3. Specify the forecast cycles to be post-processed
 - startdate*: YYYYMMDDHH of forecast cycle
 - fhr*: first forecast hour
 - lastfhr*: last forecast hour
 - incrementhr*: increment (in hours) between forecast files
4. Define location of the post-processor executables
5. Link the microphysical table *\${WRFPATH}/run/ETAMP_DATA* and control file *../parm/wrf_control.parm* to the working directory
6. Set up how many domains will be post-processed
 - For runs with a single domain, use “for domain d01”.
 - For runs with multiple domains, use “for domain d01 d02 .. dnn”
7. Create namelist *itag* that will be read in by *wrfpost.exe* from stdin (unit 5). This namelist contains 4 lines:
 - i. Name of the WRF output file to be posted.
 - ii. Format of WRF model output (netcdf or binary).
 - iii. Forecast valid time (not model start time) in WRF format.
 - iv. Model name (NMM or ARW).
8. Run *wrfpost* and check for errors.
 - The execution command in the distributed scripts is for a single processor:
wrfpost.exe < itag > outpost.
 - To run *wrfpost* on multiple processors, the command line should be:
 - LINUX-MPI systems: *mpirun -np N wrfpost.exe < itag > outpost*
 - IBM: *mpirun.lsf wrfpost.exe < itag > outpost*
9. Set up grid to post to (see full description under “Run *copygb*” below)
 - copygb* is run with a pre-defined AWIPS grid
 - gridno*: standard AWIPS grid to interpolate WRF model output to
 - copygb* ingests a kgds definition on the command line
 - copygb* ingests the contents of file *copygb_gridnav.txt* or *copygb_hwrf.txt* through variable *nav*
10. Run *copygb* and check for errors.
 - copygb.exe -xg“grid [kgds]” input_file output_file*
 - where *grid* refers to the output grid to which the native forecast is being interpolated.

The output grid can be specified in three ways:

- i. As the grid id of a pre-defined AWIPS grid:
`copygb.exe -g${gridno} -x input_file output_file`

For example, using grid 218:

```
copygb.exe -xg"218" WRFPRS_${domain}.${fhr} wrfprs_${domain}.${fhr}
```

- ii. As a user defined standard grid, such as for grid 255:
`copygb.exe -xg"255 kgds" input_file output_file`

where the user defined grid is specified by a full set of kgds parameters determining a GRIB GDS (grid description section) in the **W3fi63** format. Details on how to specify the kgds parameters are documented in file *lib/w3lib/w3fi71.f*. For example:

```
copygb.exe -xg" 255 3 109 91 37719 -77645 8 -71000 10433 9966 0 64
42000 42000" WRFPRS_${domain}.${fhr} wrfprs_${domain}.${fhr}
```

- iii. Specifying output grid as a file: When WRF-NMM output in netCDF format is processed by *wrfpost*, two text files *copygb_gridnav.txt* and *copygb_hwrf.txt* are created. These files contain the GRID GDS of a Lambert Conformal Grid (file *copygb_gridnav.txt*) or lat/lon grid (*copygb_hwrf.txt*) similar in domain and grid spacing to the one used to run the WRF-NMM model. The contents of one of these files are read into variable *nav* and can be used as input to *copygb.exe*.

```
copygb.exe -xg"$nav" input_file output_file
```

For example, when using "*copygb_gridnav.txt*" for an application the steps include:

```
read nav < 'copygb_gridnav.txt'
export nav
copygb.exe -xg"${nav}" WRFPRS_${domain}.${fhr}
wrfprs_${domain}.${fhr}
```

If scripts *run_wrfpostandgrads* or *run_wrfpostandgempak* are used, additional steps are taken to create image files (see **Visualization** section below).

Upon a successful run, *wrfpost* and *copygb* will generate output files *WRFPRS_dnn.hh* and *wrfprs_dnn.hh*, respectively, in the post-processor working directory, where "*nn*" refers to the domain id and "*hh*" denotes the forecast hour. In addition, the script *run_wrfpostandgrads* will produce a suite of gif images named *variablehh_GrADS.gif*, and the script *run_wrfpostandgempak* will produce a suite of gif images named *variablehh.gif*. An additional file containing native grid navigation information (*griddef.out*), which is currently not used, will also be produced.

If the run did not complete successfully, a log file in the post-processor working directory called *wrfpost_dnn.hh.out*, where "*nn*" is the domain id and "*hh*" is the forecast hour,

may be consulted for further information.

It should be noted that *copygb* is a flexible program that can accept several command line options specifying details of how the horizontal interpolation from the native grid to the output grid should be performed. Complete documentation of *copygb* can be found in *WPPV3/sorc/copygb/copygb.doc*.

Visualization with WPP

GEMPAK

The GEMPAK utility *nagrib* is able to decode GRIB files whose navigation is on any non-staggered grid. Hence, GEMPAK is able to decode GRIB files generated by the WRF Postprocessing package and plot horizontal fields or vertical cross sections.

A sample script named *run_wrfpostandgempak*, which is included in the *scripts* directory of the tar file, can be used to run *wrfpost*, *copygb*, and plot the following fields using GEMPAK:

- *Sfcmap_dnn_hh.gif*: mean SLP and 6 hourly precipitation
- *PrecipType_dnn_hh.gif*: precipitation type (just snow and rain)
- *850mbRH_dnn_hh.gif*: 850 mb relative humidity
- *850mbTempandWind_dnn_hh.gif*: 850 mb temperature and wind vectors
- *500mbHandVort_dnn_hh.gif*: 500 mb geopotential height and vorticity
- *250mbWindandH_dnn_hh.gif*: 250 mb wind speed isotacs and geopotential height

This script can be modified to customize fields for output. GEMPAK has an online users guide at

http://www.unidata.ucar.edu/software/gempak/help_and_documentation/manual/.

In order to use the script *run_wrfpostandgempak*, it is necessary to set the environment variable *GEMEXEC* to the path of the GEMPAK executables. For example,

```
setenv GEMEXEC /usr/local/gempak/bin
```

Note: For GEMPAK, the precipitation accumulation period for WRF-NMM is given by the variable *incrementthr* in the *run_wrfpostandgempak* script.

GrADS

The GrADS utilities *grib2ctl.pl* and *gribmap* are able to decode GRIB files whose navigation is on any non-staggered grid. These utilities and instructions on how to use them to generate GrADS control files are available from:

<http://www.cpc.ncep.noaa.gov/products/wesley/grib2ctl.html>.

The GrADS package is available from: <http://grads.iges.org/grads/grads.html>. GrADS has an online User's Guide at: <http://grads.iges.org/grads/gadoc/> and a list of basic commands for GrADS can be found at: http://grads.iges.org/grads/gadoc/reference_card.pdf.

A sample script named *run_wrfpostandgrads*, which is included in the *scripts* directory of the WRF Postprocessing package, can be used to run *wrfpost*, *copygb*, and plot the following fields using GrADS:

- *Sfcmaphh_dnn_GRADS.gif*: mean SLP and 6-hour accumulated precipitation.
- *850mbRHhh_dnn_GRADS.gif*: 850 mb relative humidity
- *850mbTempandWindhh_dnn_GRADS.gif*: 850 mb temperature and wind vectors
- *500mbHandVorthh_dnn_GRADS.gif*: 500 mb geopotential heights and absolute vorticity
- *250mbWindandHhh_dnn_GRADS.gif*: 250 mb wind speed isotacs and geopotential heights

In order to use the script *run_wrfpostandgrads*, it is necessary to:

1. Set the environmental variable *GADDIR* to the path of the GrADS fonts and auxiliary files. For example,

```
setenv GADDIR /usr/local/grads/data
```

2. Add the location of the GrADS executables to the *PATH*. For example

```
setenv PATH /usr/local/grads/bin:$PATH
```

3. Link script *cbar.gs* to the post-processor working directory. (This script is provided in WPP package, and the *run_wrfpostandgrads* script makes a link from *scripts/* to *postprd/*.) To generate the plots above, GrADS script *cbar.gs* is invoked. This script can also be obtained from the GrADS library of scripts at <http://grads.iges.org/grads/gadoc/library.html>.

Note: For GrADS, the precipitation accumulation period for WRF-NMM is plotted over the subintervals of the *tprec* hour (set in *namelist.input*).

Fields produced by *wrfpost*

Table 3 lists basic and derived fields that are currently produced by *wrfpost*. The abbreviated names listed in the second column describe how the fields should be entered in the control file (*wrf_cntrl.parm*).

Table 3: Fields produced by *wrfpost* (column 1), abbreviated names used in the *wrf_cntrl.parm* file (column 2), corresponding GRIB identification number for the field (column 3), and corresponding GRIB identification number for the vertical coordinate (column 4).

Field name	Name in control file	Grib ID	Vertical level
Radar reflectivity on model surface	RADAR REFL MDL SFCS	211	109
Pressure on model surface	PRESS ON MDL SFCS	1	109
Height on model surface	HEIGHT ON MDL SFCS	7	109
Temperature on model surface	TEMP ON MDL SFCS	11	109
Potential temperature on model surface	POT TEMP ON MDL SFCS	13	109
Dew point temperature on model surface	DWPT TEMP ON MDL SFC	17	109
Specific humidity on model surface	SPEC HUM ON MDL SFCS	51	109
Relative humidity on model surface	REL HUM ON MDL SFCS	52	109
Moisture convergence on model surface	MST CNVG ON MDL SFCS	135	109
U component wind on model surface	U WIND ON MDL SFCS	33	109
V component wind on model surface	V WIND ON MDL SFCS	34	109
Cloud water on model surface	CLD WTR ON MDL SFCS	153	109
Cloud ice on model surface	CLD ICE ON MDL SFCS	58	109
Rain on model surface	RAIN ON MDL SFCS	170	109
Snow on model surface	SNOW ON MDL SFCS	171	109
Cloud fraction on model surface	CLD FRAC ON MDL SFCS	71	109
Omega on model surface	OMEGA ON MDL SFCS	39	109
Absolute vorticity on model surface	ABS VORT ON MDL SFCS	41	109
Geostrophic streamfunction on model surface	STRMFUNC ON MDL SFCS	35	109
Turbulent kinetic energy on model surface	TRBLNT KE ON MDL SFC	158	109
Richardson number on model surface	RCHDSN NO ON MDL SFC	254	109
Master length scale on model surface	MASTER LENGTH SCALE	226	109
Asymptotic length scale on model surface	ASYMPT MSTR LEN SCL	227	109
Radar reflectivity on pressure surface	RADAR REFL ON P SFCS	211	100

Height on pressure surface	HEIGHT OF PRESS SFCS	7	100
Temperature on pressure surface	TEMP ON PRESS SFCS	11	100
Potential temperature on pressure surface	POT TEMP ON P SFCS	13	100
Dew point temperature on pressure surface	DWPT TEMP ON P SFCS	17	100
Specific humidity on pressure surface	SPEC HUM ON P SFCS	51	100
Relative humidity on pressure surface	REL HUMID ON P SFCS	52	100
Moisture convergence on pressure surface	MST CNVG ON P SFCS	135	100
U component wind on pressure surface	U WIND ON PRESS SFCS	33	100
V component wind on pressure surface	V WIND ON PRESS SFCS	34	100
Omega on pressure surface	OMEGA ON PRESS SFCS	39	100
Absolute vorticity on pressure surface	ABS VORT ON P SFCS	41	100
Geostrophic streamfunction on pressure surface	STRMFUNC ON P SFCS	35	100
Turbulent kinetic energy on pressure surface	TRBLNT KE ON P SFCS	158	100
Cloud water on pressure surface	CLOUD WATR ON P SFCS	153	100
Cloud ice on pressure surface	CLOUD ICE ON P SFCS	58	100
Rain on pressure surface	RAIN ON P SFCS	170	100
Snow water on pressure surface	SNOW ON P SFCS	171	100
Total condensate on pressure surface	CONDENSATE ON P SFCS	135	100
Mesinger (Membrane) sea level pressure	MESINGER MEAN SLP	130	102
Shuell sea level pressure	SHUELL MEAN SLP	2	102
2 M pressure	SHELTER PRESSURE	1	105
2 M temperature	SHELTER TEMPERATURE	11	105
2 M specific humidity	SHELTER SPEC HUMID	51	105
2 M dew point temperature	SHELTER DEWPOINT	17	105
2 M RH	SHELTER REL HUMID	52	105
10 M u component wind	U WIND AT ANEMOM HT	33	105
10 M v component wind	V WIND AT ANEMOM HT	34	105
10 M potential temperature	POT TEMP AT 10 M	13	105
10 M specific humidity	SPEC HUM AT 10 M	51	105
Surface pressure	SURFACE PRESSURE	1	1
Terrain height	SURFACE HEIGHT	7	1
Skin potential temperature	SURFACE POT TEMP	13	1

Skin specific humidity	SURFACE SPEC HUMID	51	1
Skin dew point temperature	SURFACE DEWPOINT	17	1
Skin Relative humidity	SURFACE REL HUMID	52	1
Skin temperature	SFC (SKIN) TEMPRATUR	11	1
Soil temperature at the bottom of soil layers	BOTTOM SOIL TEMP	85	111
Soil temperature in between each of soil layers	SOIL TEMPERATURE	85	112
Soil moisture in between each of soil layers	SOIL MOISTURE	144	112
Snow water equivalent	SNOW WATER EQUIVALNT	65	1
Snow cover in percentage	PERCENT SNOW COVER	238	1
Heat exchange coeff at surface	SFC EXCHANGE COEF	208	1
Vegetation cover	GREEN VEG COVER	87	1
Soil moisture availability	SOIL MOISTURE AVAIL	207	112
Ground heat flux - instantaneous	INST GROUND HEAT FLX	155	1
Lifted index—surface based	LIFTED INDEX—SURFCE	131	101
Lifted index—best	LIFTED INDEX—BEST	132	116
Lifted index—from boundary layer	LIFTED INDEX—BNDLYR	24	116
CAPE	CNVCT AVBL POT ENRGY	157	1
CIN	CNVCT INHIBITION	156	1
Column integrated precipitable water	PRECIPITABLE WATER	54	200
Column integrated cloud water	TOTAL COLUMN CLD WTR	136	200
Column integrated cloud ice	TOTAL COLUMN CLD ICE	137	200
Column integrated rain	TOTAL COLUMN RAIN	138	200
Column integrated snow	TOTAL COLUMN SNOW	139	200
Column integrated total condensate	TOTAL COL CONDENSATE	140	200
Helicity	STORM REL HELICITY	190	106
U component storm motion	U COMP STORM MOTION	196	106
V component storm motion	V COMP STORM MOTION	197	106
Accumulated total precipitation	ACM TOTAL PRECIP	61	1
Accumulated convective precipitation	ACM CONVCTIVE PRECIP	63	1
Accumulated grid-scale precipitation	ACM GRD SCALE PRECIP	62	1
Accumulated snowfall	ACM SNOWFALL	65	1
Accumulated total snow melt	ACM SNOW TOTAL MELT	99	1

Precipitation type (4 types) - instantaneous	INSTANT PRECIP TYPE	140	1
Precipitation rate - instantaneous	INSTANT PRECIP RATE	59	1
Composite radar reflectivity	COMPOSITE RADAR REFL	212	200
Low level cloud fraction	LOW CLOUD FRACTION	73	214
Mid level cloud fraction	MID CLOUD FRACTION	74	224
High level cloud fraction	HIGH CLOUD FRACTION	75	234
Total cloud fraction	TOTAL CLD FRACTION	71	200
Time-averaged total cloud fraction	AVG TOTAL CLD FRAC	71	200
Time-averaged stratospheric cloud fraction	AVG STRAT CLD FRAC	213	200
Time-averaged convective cloud fraction	AVG CNVCT CLD FRAC	72	200
Cloud bottom pressure	CLOUD BOT PRESSURE	1	2
Cloud top pressure	CLOUD TOP PRESSURE	1	3
Cloud bottom height (above MSL)	CLOUD BOTTOM HEIGHT	7	2
Cloud top height (above MSL)	CLOUD TOP HEIGHT	7	3
Convective cloud bottom pressure	CONV CLOUD BOT PRESS	1	242
Convective cloud top pressure	CONV CLOUD TOP PRESS	1	243
Shallow convective cloud bottom pressure	SHAL CU CLD BOT PRES	1	248
Shallow convective cloud top pressure	SHAL CU CLD TOP PRES	1	249
Deep convective cloud bottom pressure	DEEP CU CLD BOT PRES	1	251
Deep convective cloud top pressure	DEEP CU CLD TOP PRES	1	252
Grid scale cloud bottom pressure	GRID CLOUD BOT PRESS	1	206
Grid scale cloud top pressure	GRID CLOUD TOP PRESS	1	207
Convective cloud fraction	CONV CLOUD FRACTION	72	200
Convective cloud efficiency	CU CLOUD EFFICIENCY	134	200
Above-ground height of LCL	LCL AGL HEIGHT	7	5
Pressure of LCL	LCL PRESSURE	1	5
Cloud top temperature	CLOUD TOP TEMPS	11	3
Temperature tendency from radiative fluxes	RADFLX CNVG TMP TNDY	216	109
Temperature tendency from shortwave radiative flux	SW RAD TEMP TNDY	250	109
Temperature tendency from	LW RAD TEMP TNDY	251	109

longwave radiative flux			
Outgoing surface shortwave radiation - instantaneous	INSTN OUT SFC SW RAD	211	1
Outgoing surface longwave radiation - instantaneous	INSTN OUT SFC LW RAD	212	1
Incoming surface shortwave radiation - time-averaged	AVE INCMG SFC SW RAD	204	1
Incoming surface longwave radiation - time-averaged	AVE INCMG SFC LW RAD	205	1
Outgoing surface shortwave radiation - time-averaged	AVE OUTGO SFC SW RAD	211	1
Outgoing surface longwave radiation - time-averaged	AVE OUTGO SFC LW RAD	212	1
Outgoing model top shortwave radiation - time-averaged	AVE OUTGO TOA SW RAD	211	8
Outgoing model top longwave radiation - time-averaged	AVE OUTGO TOA LW RAD	212	8
Incoming surface shortwave radiation - instantaneous	INSTN INC SFC SW RAD	204	1
Incoming surface longwave radiation - instantaneous	INSTN INC SFC LW RAD	205	1
Roughness length	ROUGHNESS LENGTH	83	1
Friction velocity	FRICION VELOCITY	253	1
Surface drag coefficient	SFC DRAG COEFFICIENT	252	1
Surface u wind stress	SFC U WIND STRESS	124	1
Surface v wind stress	SFC V WIND STRESS	125	1
Surface sensible heat flux - time-averaged	AVE SFC SENHEAT FX	122	1
Ground heat flux - time-averaged	AVE GROUND HEAT FX	155	1
Surface latent heat flux - time-averaged	AVE SFC LATHEAT FX	121	1
Surface momentum flux - time-averaged	AVE SFC MOMENTUM FX	172	1
Accumulated surface evaporation	ACC SFC EVAPORATION	57	1
Surface sensible heat flux - instantaneous	INST SFC SENHEAT FX	122	1
Surface latent heat flux - instantaneous	INST SFC LATHEAT FX	121	1
Latitude	LATITUDE	176	1
Longitude	LONGITUDE	177	1
Land sea mask (land=1, sea=0)	LAND SEA MASK	81	1
Sea ice mask	SEA ICE MASK	91	1
Surface midday albedo	SFC MIDDAY ALBEDO	84	1
Sea surface temperature	SEA SFC TEMPERATURE	80	1
Press at tropopause	PRESS AT TROPOPAUSE	1	7

Temperature at tropopause	TEMP AT TROPOPAUSE	11	7
Potential temperature at tropopause	POTENTL TEMP AT TROP	13	7
U wind at tropopause	U WIND AT TROPOPAUSE	33	7
V wind at tropopause	V WIND AT TROPOPAUSE	34	7
Wind shear at tropopause	SHEAR AT TROPOPAUSE	136	7
Height at tropopause	HEIGHT AT TROPOPAUSE	7	7
Temperature at flight levels	TEMP AT FD HEIGHTS	11	103
U wind at flight levels	U WIND AT FD HEIGHTS	33	103
V wind at flight levels	V WIND AT FD HEIGHTS	34	103
Freezing level height (above mean sea level)	HEIGHT OF FRZ LVL	7	4
Freezing level RH	REL HUMID AT FRZ LVL	52	4
Highest freezing level height	HIGHEST FREEZE LVL	7	204
Pressure in boundary layer (30 mb mean)	PRESS IN BNDRY LYR	1	116
Temperature in boundary layer (30 mb mean)	TEMP IN BNDRY LYR	11	116
Potential temperature in boundary layers (30 mb mean)	POT TMP IN BNDRY LYR	13	116
Dew point temperature in boundary layer (30 mb mean)	DWPT IN BNDRY LYR	17	116
Specific humidity in boundary layer (30 mb mean)	SPC HUM IN BNDRY LYR	51	116
RH in boundary layer (30 mb mean)	REL HUM IN BNDRY LYR	52	116
Moisture convergence in boundary layer (30 mb mean)	MST CNV IN BNDRY LYR	135	116
Precipitable water in boundary layer (30 mb mean)	P WATER IN BNDRY LYR	54	116
U wind in boundary layer (30 mb mean)	U WIND IN BNDRY LYR	33	116
V wind in boundary layer (30 mb mean)	V WIND IN BNDRY LYR	34	116
Omega in boundary layer (30 mb mean)	OMEGA IN BNDRY LYR	39	116
Visibility	VISIBILITY	20	1
Vegetation type	VEGETATION TYPE	225	1
Soil type	SOIL TYPE	224	1
Canopy conductance	CANOPY CONDUCTANCE	181	1
PBL height	PBL HEIGHT	221	1

Slope type	SLOPE TYPE	222	1
Snow depth	SNOW DEPTH	66	1
Liquid soil moisture	LIQUID SOIL MOISTURE	160	112
Snow free albedo	SNOW FREE ALBEDO	170	1
Maximum snow albedo	MAXIMUM SNOW ALBEDO	159	1
Canopy water evaporation	CANOPY WATER EVAP	200	1
Direct soil evaporation	DIRECT SOIL EVAP	199	1
Plant transpiration	PLANT TRANSPIRATION	210	1
Snow sublimation	SNOW SUBLIMATION	198	1
Air dry soil moisture	AIR DRY SOIL MOIST	231	1
Soil moist porosity	SOIL MOIST POROSITY	240	1
Minimum stomatal resistance	MIN STOMATAL RESIST	203	1
Number of root layers	NO OF ROOT LAYERS	171	1
Soil moist wilting point	SOIL MOIST WILT PT	219	1
Soil moist reference	SOIL MOIST REFERENCE	230	1
Canopy conductance - solar component	CANOPY COND SOLAR	246	1
Canopy conductance - temperature component	CANOPY COND TEMP	247	1
Canopy conductance - humidity component	CANOPY COND HUMID	248	1
Canopy conductance - soil component	CANOPY COND SOILM	249	1
Potential evaporation	POTENTIAL EVAP	145	1
Heat diffusivity on sigma surface	DIFFUSION H RATE S S	182	107
Surface wind gust	SFC WIND GUST	180	1
Convective precipitation rate	CONV PRECIP RATE	214	1
Radar reflectivity at certain above ground heights	RADAR REFL AGL	211	105

RIP4

RIP Introduction

RIP (which stands for Read/Interpolate/Plot) 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 can also be used to visualize model input or analyses on model grids.

RIP has been under continuous development since 1991, primarily by Mark Stoelinga at both NCAR and the University of Washington. 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 both the WRF-NMM and WRF-ARW modeling systems.

It is strongly recommended that users read the complete RIP User's Guide found at: <http://www.mmm.ucar.edu/wrf/users/docs/ripug.pdf>. A condensed version is given here for a quick reference.

RIP Software Requirements

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 (preferably 4.0 or higher). The NetCDF library is also required for I/O.

RIP Environment Settings

An important environment variable for the RIP system is ***RIP_ROOT***. ***RIP_ROOT*** should be assigned the path name of the directory where all the RIP program and utility files (*color.tbl*, *stationlist*, *.ascii* files, *psadilookup.dat*) will reside. The natural choice for the ***RIP_ROOT*** directory is the ***RIP*** subdirectory that is created after the RIP tar file is unpacked. For simplicity, define ***RIP_ROOT*** in one of the UNIX start-up files. For example, add the following to the shell configuration file (such as **.login** or **.cshrc**):

```
setenv RIP_ROOT /path-to/RIP
```

Obtaining the RIP Code

The RIP4 package can be downloaded from:
<http://www.dtcenter.org/wrf-nmm/users/downloads>.

Once the *tar* file is obtained, *gunzip* and *untar* the file.

```
tar -zxvf RIP4_v4.4.TAR.gz
```

This command will create a directory called **RIP**.

RIP Directory Structure

Under the main directory of **RIP** reside the following files and subdirectories:

:

- **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*).
- **Makefile**, the top-level make file used to compile and link RIP.
- **README**, a text file containing basic information on running RIP.
- **color.tbl**, a file that contains a table defining the colors you want to have available for RIP plots.
- **eta_micro_lookup.dat**, a file that contains "look-up" table data for the Ferrier microphysics scheme.
- **psadillookup.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. These files include *bwave.in*, *grav2d_x.in*, *hill2d.in*, *qss.in*, *rip_sample.in*, *ripdp_wrfarw_sample.in*, *ripdp_wrfnm_sample.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.

Installing the RIP Code

RIP uses a build mechanism similar to that used by the WRF model. First issue the **configure** command, followed by the **compile** command.

```
./configure
```

Choose one of the configure options listed. Check the **configure.rip** file created and edit for compile options/paths, if necessary.

```
./compile >& compile_rip.log &
```

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

To remove all built files, as well as the *configure.rip*, type:

clean

This action is recommended if a mistake is made during the installation process.

RIP Functionalities

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 CGM file is created, which can be viewed with any one of several different metacode translator applications. 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.
- 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.

RIP Data Preparation (RIPDP)

RIP does not ingest model output files directly. First, a preprocessing step, RIPDP (which stands for RIP Data Preparation), must be executed which converts the model output data

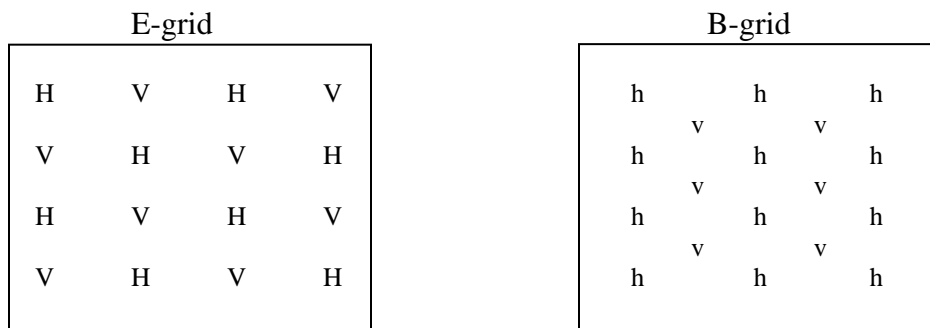
files to RIP-format data files. The primary difference between these two types of files is that model output is in NetCDF or GRIB format and may contain all times and all variables in a single file (or a few files), whereas RIP data has each variable at each time in a separate file in binary format.

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 *make*, it should produce executable programs called *ripdp_mm5*, *ripdp_wrfarw*, and *ripdp_wrfnmm*. Although these are three separate programs, they serve the same purpose, and only *ripdp_wrfnmm* will be described here.

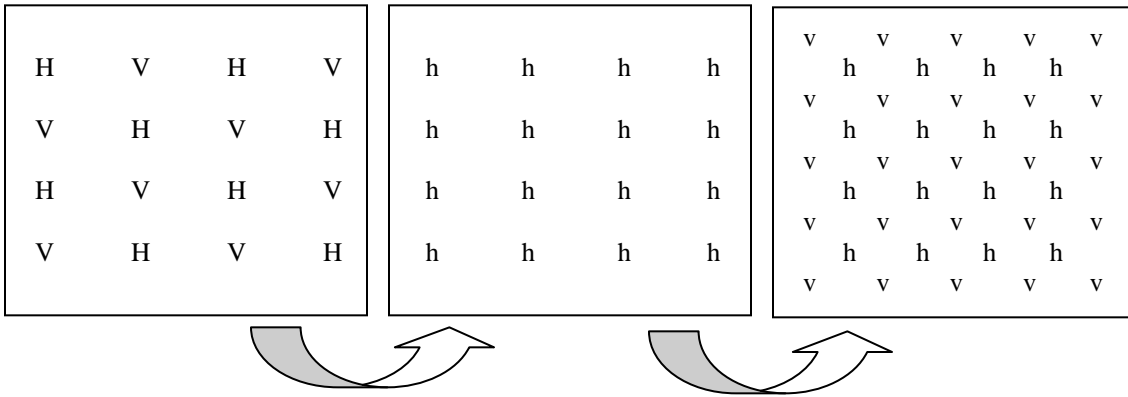
The WRF-NMM model uses a rotated latitude/longitude projection on the E-grid, both of which introduce special challenges for processing and plotting WRF-NMM data. RIPDP and RIP have been modified to handle the rotated latitude/longitude projection, however, the grid staggering in WRF-NMM requires additional data processing.

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.



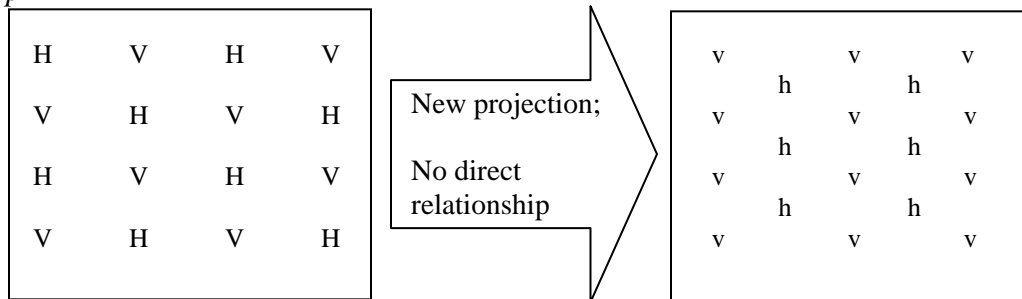
In the first method (*iinterp=0*), we define a B-grid in which its mass (h) points collocate with all the mass (H) and velocity (V) points in the E-grid, and the B-grid's velocity (v) points are staggered in the usual B-grid way (see illustration below). The RIPDP-created data files retain only the E-grid data, but then when they are ingested into RIP, the E-grid H-point data are transferred directly to overlapping B-grid mass points, and non-overlapping B-grid mass points and all velocity 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.

$iinterp = 0$



The second method ($iinterp=1$) 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.

$iinterp=1$



It should be noted that if $iinterp=1$ is set, 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.

RIPDP Namelist

The namelist file for RIPDP contains the namelist *&userin*. All of the *&userin* namelist variables are listed below. Each variable has a default value, which is the value this variable will take if its specification is omitted from the namelist. Additional details for each namelist variable can be found in [Chapter 3](#) of the full RIP User's Guide.

Variable Name	Default Value	Description
<i>ptimes</i>	9.0E+09	Times to process. This can be a string of times or a series in the form of <i>A,-B,C</i> , which is interpreted as "times from hour <i>A</i> to hour <i>B</i> , every <i>C</i> hours".
<i>ptimeunits</i>	'h'	Units of <i>ptimes</i> . This can be either 'h' (for hours), 'm' (for minutes), or 's' (for seconds).
<i>iptimes</i>	99999999	Times to process in the form of 8-digit "mdate" times (i.e. YYMMDDHH). A value of 99999999 indicates <i>ptimes</i> is being used instead.
<i>tacc</i>	1.0	Time tolerance, in seconds. Any times encountered in the model output that are within <i>tacc</i> seconds of one of the times specified in <i>ptimes</i> or <i>iptimes</i> will be processed.
<i>discard</i>	''	Names of variables that, if encountered in the model data file, will not be processed.
<i>retain</i>	''	Names of variables that, if encountered in the model data file, <u>should</u> be processed, even though the user specified <i>basic</i> on the <i>ripdp_wrfnm</i> command line.
<i>iinterp</i>	0	NMM Only: Method for defining B-grid (described above) 0 = Collocated high-density B-grid 1 = Interpolate to a new B-grid

If *iinterp=1* then the following namelist variables must also be set for the indicated domain on which the new B-grid will be based.

Variable Name	Default Value	Description
<i>dskmcib</i>	50.0	Grid spacing, in km, of the centered domain.
<i>miycorsib</i> , <i>mjxcorsib</i>	100	Grid points in the <i>y</i> and <i>x</i> directions,

		respectively, of the centered domain.
<i>nprojib</i>	1	Map projection number (0: none/ideal, 1: LC, 2: PS, 3: ME, 4: SRCE) of the centered domain.
<i>xlatcib, xloncib</i>	45.0, -90.0	Central latitude and longitude, respectively, for the centered domain.
<i>truelat1ib, truelat2ib</i>	30.0, 60.0	Two true latitudes for the centered domain.
<i>miyib, mjxib</i>	75	Number of grid points in the y and x directions, respectively, of the fine domain.
<i>yicornib, xjcornib</i>	25	Centered domain y and x locations, respectively, of the lower left corner point of the fine domain.
<i>dskmib</i>	25.0	Grid spacing, in km, of the fine domain.

An example of a namelist input file for *ripdp_wrfnm*, called *ripdp_wrfnm_sample.in*, is provided in the RIP tar file in the *sample_infiles* directory.

Running RIPDP

RIPDP has the following usage:

```
ripdp_wrfnm -n namelist_file model-data-set-name [basic|all] data_file_1
data_file_2 data_file_3 ...
```

In order to provide the user more control when processing the data, a namelist needs to be specified by means of the *-n* option, with *namelist_file* specifying the path name of the file containing the namelist.

The argument *model-data-set-name* can be any string you choose, that uniquely defines the model output data set. It will be used in the file names of all the new RIP data files that are created.

The *basic* option causes *ripdp_wrfnm* to process only the basic variables that RIP requires, whereas *all* causes *ripdp_wrfnm* to process all variables encountered in the model output file. It produces files for those variables using the variable name provided in the model output to create the file name. If *all* is specified, the *discard* variable can be used in the RIPDP namelist to prevent processing of unwanted variables. 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.

Finally, *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. If model output exists for nested domains, RIPDP must be run for each domain separately and each run must be given a new *model-data-set-name*.

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 data that will be accessed by RIP to produce plots. See [Appendix C](#) in the full RIP user’s guide for a description of how these files are named and the format of their contents.

RIP User Input File (UIF)

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.

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

The first namelist in the UIF is called *&userin*. A description of each variable is shown below. Each variable has a default value, which is the value this variable will take if its specification is omitted from the namelist. Additional details for each namelist variable can be found in [Chapter 4](#) of the full RIP User’s Guide.

Variable Name	Default Value	Description
<i>idotitle, title</i>	1, 'auto'	Control the first part of the first title line.
<i>titlecolor</i>	'def.foreground'	Specifies the color for the text in the title.
<i>iinittime</i>	1	If flag = 1, the initial date and time (in UTC) will be printed in the title.
<i>ifcstime</i>	1	If flag = 1, the forecast lead time (in hours) will be printed in the title.
<i>ivalidtime</i>	1	If flag = 1, the valid date and time (in both UTC and local time) will be printed in the title.
<i>inearesth</i>	0	If flag = 1, plot valid time as two digits rather than 4 digits.
<i>timezone</i>	-7.0	Offset from Greenwich time (UTC) for the local time zone.
<i>iusedaylightrule</i>	1	If flag = 1, apply daylight saving rule.
<i>ptimes</i>	9.0E+09	Times to process. This can be a string of times or a series in the form of A,-B,C, which is interpreted as "times from hour A to hour B, every C hours".
<i>ptimeunits</i>	' h'	Units of <i>ptimes</i> .

		This can be either 'h' (for hours), 'm' (for minutes), or 's' (for seconds).
<i>iptimes</i>	99999999	Times to process in the form of 8-digit "mdate" times (i.e. YYMMDDHH). A value of 99999999 indicates <i>ptimes</i> is being used instead.
<i>tacc</i>	1.0	Time tolerance, in seconds. Any times encountered in the model output that are within <i>tacc</i> seconds of one of the times specified in <i>ptimes</i> or <i>iptimes</i> will be processed.
<i>flmin, frmax, fbmin, and ftmax</i>	0.05,0.95,0.10,0.90	Left frame limit, right frame limit, bottom frame limit, and top frame limit, respectively.
<i>ntextq</i>	0	Text quality specifier [0=high; 1=medium; 2=low].
<i>ntextcd</i>	0	Text font specifier [0=complex (Times); 1=duplex (Helvetica)].
<i>fcoffset</i>	0.0	Change initial time to something other than output initial time.
<i>idotser</i>	0	If flag = 1, generate time series ASCII output files (no plots).
<i>idescriptive</i>	1	If flag = 1, use more descriptive plot titles.
<i>icgmsplit</i>	0	If flag = 1, split metacode into several files.
<i>maxfld</i>	10	Reserve memory for RIP.
<i>itrajcalc</i>	0	If flag = 1, turns on trajectory calculation (use &trajcalc namelist as well).
<i>imakev5d</i>	0	If flag = 1, generates output for Vis5D.
<i>rip_root</i>	'/dev/null'	Over-ride the path name specified in UNIX environment variable RIP_ROOT .

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 the [Calculating and Plotting Trajectories with RIP](#) section and in [Chapter 6](#) of the full RIP User's Guide.

b. The plot specification table

The plot specification table (PST) provides 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 indicates a PST. Following that there 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. All the PSSs within a PSL must be separated by semicolons, 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 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](#) in the full RIP User's Guide.

Each keyword has a variable associated with it in the program, and this variable may be of type integer, real, character, logical, or array. The keywords that are associated with a

real variable expect values that are of Fortran floating point format. The keywords that are associated with an integer variable also expect values that are of Fortran floating point format because they are initially read in as a floating point number, and then rounded (not truncated) to the nearest integer. 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 as `.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. The values should be separated by commas, as mentioned above.

All keywords are set to a default value prior to the reading of the PST. 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 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 is reset to its default value after every PSL. Keywords that remember 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 section has described the basic rules to follow in creating the PST. [Appendix A](#) in the full RIP User's Guide provides a description of all of the available keywords, in alphabetical order.

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 sections, 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*. The *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. Again, if nested domains were run, *rip* will be run for each domain separately. The *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.cgm*. If separate files were requested for each plot time, they are named *rip-execution-name.cgmA*, *rip-execution-name.cgmB*, etc.

Although the 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*.

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 (*.cgm* files) in that directory, and a subdirectory called *data* that contains the large number of RIP data files

Calculating and Plotting Trajectories with RIP

Because trajectories are a unique feature of RIP and require special instructions to create, this section is devoted to a general explanation of the trajectory calculation 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:

Variable Name	Description
<i>rtim</i>	The release time (in forecast hours) for the trajectories.
<i>ctim</i>	The completion time (in forecast hours) for the trajectories. Note: If <i>rtim</i> < <i>ctim</i> , trajectories are forward. If <i>rtim</i> > <i>ctim</i> , trajectories are backward.
<i>dtfile</i>	The time increment (in seconds) between data files.
<i>dttraj</i>	The time step (in seconds) for trajectory calculation.
<i>vctray</i>	The vertical coordinate of values specified for <i>zktraj</i> . 's': <i>zktraj</i> values are model vertical level indices

	<p>'p': <i>zktraj</i> values are pressure values, in mb 'z': <i>zktraj</i> values are height values, in km 'm': <i>zktraj</i> values are temperature values, in C 't': <i>zktraj</i> values are potential temperature values, in K 'e': <i>zktraj</i> values are equivalent potential temperature values, in K</p>
<i>ihydrometeor</i>	If flag=1, trajectory calculation algorithm uses the hydrometeor fall speed instead of the vertical air velocity.
<i>xjtraj,yitraj</i>	<i>x</i> and <i>y</i> values (in grid points) of the initial positions of the trajectories.
<i>zktraj</i>	Vertical location of the initial points of the trajectories.

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 trajectory 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 PST is left blank (except that the first three lines comprising the PST banner are retained). If diagnostic quantities are desired, they can be requested in the PST (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 PST. 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](#) of the full RIP User's Guide can be calculated along trajectories, with the exception of the Sawyer-Eliassen diagnostics. Each desired diagnostic quantity should be specified in its own 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.

Once the input file is set up, RIP is run as outlined in the [Running RIP](#) section. Since no plots are generated when RIP is run in trajectory calculation mode, no *rip-execution-name.cgm* file is created. 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 PST.

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*. Keyword *feld* should be set to one of five possibilities: *arrow*, *ribbon*, *swarm*, *gridswarm*, or *circle* (these fields are described in detail below). Keyword *ptyp* should be set to either *ht* (for "horizontal trajectory plot") or *vt* (for "vertical (cross section) trajectory plot"). And keyword *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 grids 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 grids 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*).

See "Keywords", in [Appendix A](#) in the full RIP User's Guide for more details on optional keywords that affect trajectory plots.

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 PST 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 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*.

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.

Creating Vis5D Dataset with RIP

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 PST with only a minimum of information necessary since no plots are produced. 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](#) in the full RIP User's Guide 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 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.

Once the user input file is set up, RIP is run as outlined in the [Running RIP](#) section. Since no plots are generated when RIP is run in Vis5D mode, no *rip-execution-name.cgm* file is created. 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 be included in the Vis5D data set. Therefore, you don't have to explicitly request *feld=map* in the PST. 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 PST.