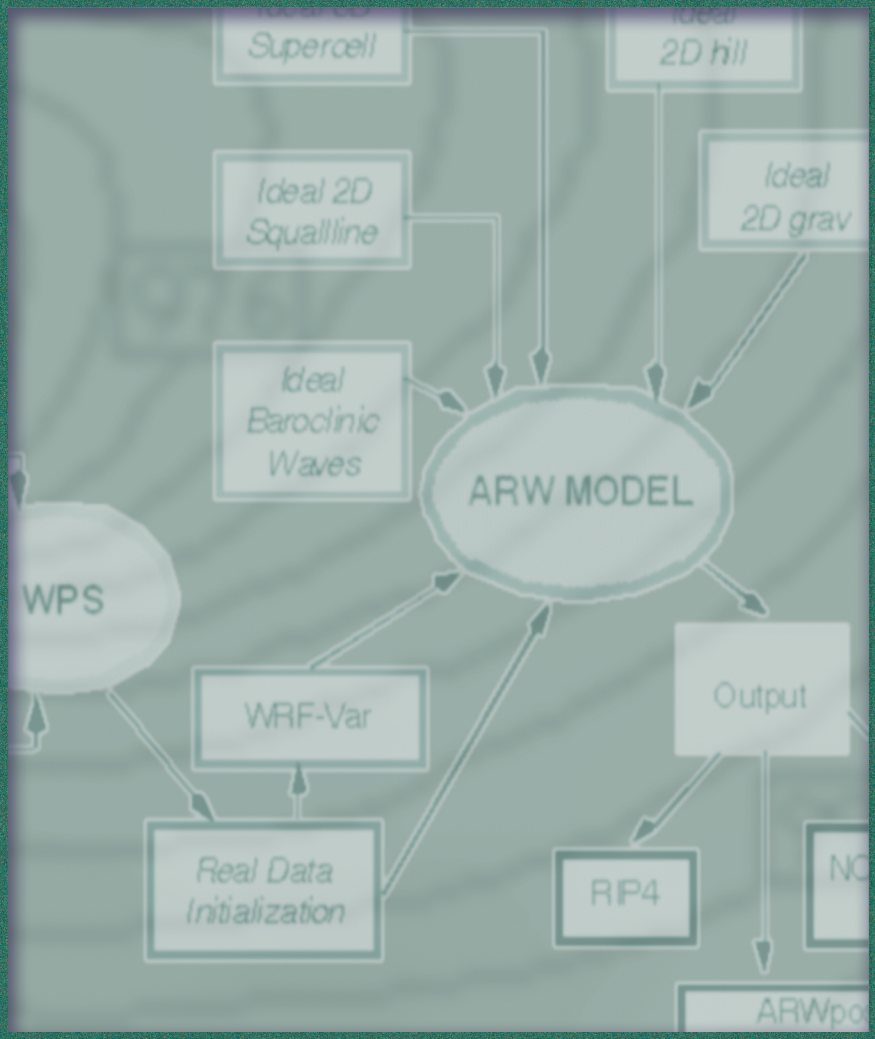
WEATHER RESEARCH & FORECASTING

ARW

Version 3 Modeling System User's Guide
April 2008



Mesoscale & Microscale Meteorology Division * National Center for Atmospheric Research

Foreword

This User's Guide describes the Advanced Research WRF (ARW) Version 3.0 modeling system, released in April 2008. As the ARW is developed further, this document will be continuously enhanced and updated.

This document is complimentary to the ARW Tech Note (http://www.mmm.ucar.edu/wrf/users/docs/arw_v2.pdf), which describes the equations, numerics, boundary conditions, and nesting etc. in greater detail. The Version 3 of the Tech Note will be available in the summer of 2008.

Highlights of updates to WRFV3 include:

- Global modeling capability;
- New physics options: Morrison 2-moment microphysics, Goddard 6-class microphysics, Asymmetric Convective Model 2 planetary boundary layer scheme and Pleim-Xu land-surface model, unified Noah LSM, new Grell scheme;
- Zaengl radiation/topography (sloping and shadowing) effects for Dudhia shortwave radiation scheme;
- Implicit upper gravity-wave absorbing layer;
- Digital filter initialization;
- New idealized cases for large eddy simulation, full-physics sea-breeze and global
- Variable time-step capability
- Merged WRF-Var and WRF in the same directory
- WRF-Var Version 3
- WRF Pre-Processing System Version 3

The Version 3 modeling system programs are not backward compatible, and it no longer supports WRFSI. For documentation of older programs, please refer to earlier versions of the User's Guide.

For the latest version of this document, please visit the ARW Users' Web site at http://www.mmm.ucar.edu/wrf/users/.

Please send feedback to wrfhelp@ucar.edu.

Contributors to this guide:
Wei Wang, Dale Barker, Cindy Bruyère, Michael Duda, Jimy Dudhia, Dave Gill, John Michalakes, and Syed Rizvi

1. Overview

2. Software Installation

3. The WRF Preprocessing System (WPS)

4. WRF Initialization

5. WRF Model

6. WRF-Var

7. WRF Software

8. Post-Processing Programs

Chapter 1: Overview

Table of Contents

Introduction

The Advanced Research WRF (ARW) modeling system has been in development for the past few years. The current release is Version 3, available since April 2008. The ARW is designed to be a flexible, state-of-the-art atmospheric simulation system that is portable and efficient on available parallel computing platforms. The ARW is suitable for use in a broad range of applications across scales ranging from meters to thousands of kilometers, including:

- Idealized simulations (e.g. LES, convection, baroclinic waves)
- Parameterization research
- Data assimilation research
- Forecast research
- Real-time NWP
- Coupled-model applications
- Teaching

The Mesoscale and Microscale Meteorology Division of NCAR is currently maintaining and supporting a subset of the overall WRF code (Version 3) that includes:

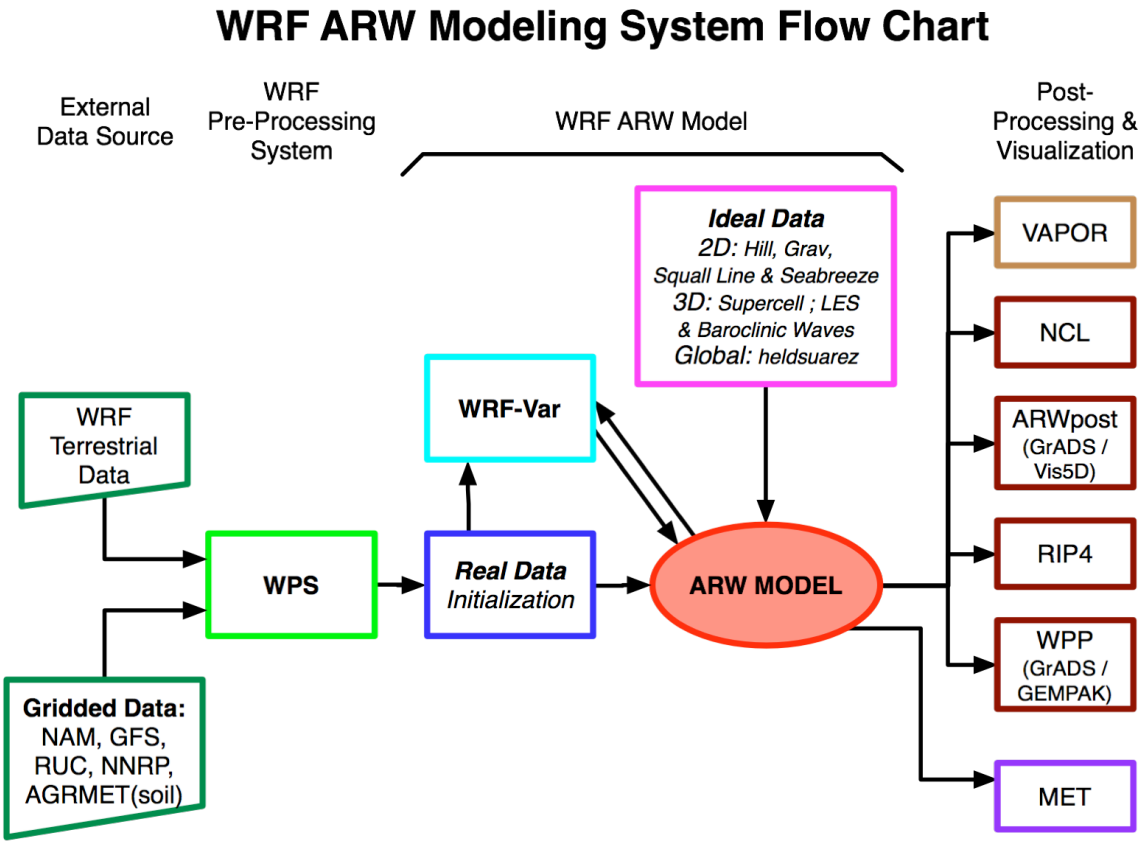
- WRF Software Framework (WSF)
- Advanced Research WRF (ARW) dynamic solver, including one-way, two-way nesting and moving nest.
- The WRF Preprocessing System (WPS)
- WRF Variational Data Assimilation (WRF-Var) system which currently supports 3DVAR capability
- Numerous physics packages contributed by WRF partners and the research community
- Several graphics programs and conversion programs for other graphics tools

And these are the subjects of this document.

The WRF modeling system software is in the public domain and is freely available for community use.

The WRF Modeling System Program Components

The following figure shows the flowchart for the WRF Modeling System Version 3.



As shown in the diagram, the WRF Modeling System consists of these major programs:

- The WRF Preprocessing System (WPS)
- WRF-Var
- ARW solver
- Post-processing & Visualization tools

WPS

This program is used primarily for real-data simulations. Its functions include 1) defining simulation domains; 2) interpolating terrestrial data (such as terrain, landuse, and soil types) to the simulation domain; and 3) degribbing and interpolating meteorological data from another model to this simulation domain. Its main features include:

- GRIB 1/2 meteorological data from various centers around the world

- Map projections for 1) polar stereographic, 2) Lambert-Conformal, 3) Mercator and 4) latitude-longitude
- Nesting
- User-interfaces to input other static data as well as met data

WRF-Var

This program is optional, but can be used to ingest observations into the interpolated analyses created by WPS. It can also be used to update WRF model's initial condition when WRF model is run in cycling mode. Its main features are as follows.

- It is based on incremental variational data assimilation technique
- Conjugate gradient method is utilized to minimized the cost function in analysis control variable space
- Analysis is performed on un-staggered Arakawa A-grid
- Analysis increments are interpolated to staggered Arakawa C-grid and it gets added to the background (first guess) to get final analysis at WRF-model grid
- Conventional observation data input may be supplied both in ASCII or "PREPBUFR" format via "obsproc" utility
- Multiple radar data (reflectivity & radial velocity) input is supplied through ASCII format
- Horizontal component of the background (first guess) error is represented via recursive filter (for regional) or power spectrum (for global). The vertical component is applied through projections on climatologically generated averaged eigenvectors and its corresponding eigenvalues
- Horizontal and vertical background errors are non-separable. Each eigen vector has its own horizontal climatologically determined length scale
- Preconditioning of background part of the cost function is done via control variable transform U defined as $B= UU^T$
- It includes "gen_be" utility to generate the climatological background error covariance estimate via the NMC-method or ensemble perturbations
- It includes a "verification" package both with respect to "observations" or "analysis"
- A utility program to update WRF boundary condition file after WRF-Var

ARW Solver

This is the key component of the modeling system, which is composed of several initialization programs for idealized, and real-data simulations, and the numerical integration program. It also includes a program to do one-way nesting. The key feature of the WRF model includes:

- fully compressible nonhydrostatic equations with hydrostatic option
- regional and global applications
- complete coriolis and curvature terms

- two-way nesting with multiple nests and nest levels
- one-way nesting
- moving nests
- mass-based terrain following coordinate
- vertical grid-spacing can vary with height
- map-scale factors for these projections:
    - polar stereographic (conformal)
    - Lambert-conformal
    - Mercator (conformal)
    - Latitude and longitude which can be rotated
- Arakawa C-grid staggering
- Runge-Kutta 2nd and 3rd order time integration options
- scalar-conserving flux form for prognostic variables
- 2nd to 6th order advection options (horizontal and vertical)
- positive-definite advection option for moisture, scalar and TKE
- time-split small step for acoustic and gravity-wave modes:
    - small step horizontally explicit, vertically implicit
    - divergence damping option and vertical time off-centering
    - external-mode filtering option
- Upper boundary aborption and Rayleigh damping
- lateral boundary conditions
    - idealized cases: periodic, symmetric, and open radiative
    - real cases: specified with relaxation zone
- full physics options for land-surface, planetary boundary layer, atmospheric and surface radiation, microphysics and cumulus convection
- grid analysis nudging and observation nudging
- digital filter initialization
- a number of idealized examples

Graphics and Verification Tools

Several programs are supported, including RIP4 (based on NCAR Graphics), NCAR Graphics Command Language (NCL), and conversion programs for other readily available graphics packages: GrADS and Vis5D.

Program VAPOR, **V**isualization and **A**nalysis **P**latform for **O**cean, Atmosphere, and Solar **R**esearchers (http://www.vapor.ucar.edu/), is a 3-dimensional data visualization tool, and it is developed and supported by the VAPOR team at NCAR (vapor@ucar.edu).

Program MET, **M**odel **E**valuation **T**ools (http://www.dtcenter.org/met/users/), is developed and supported by the Developmental Testbed Center at NCAR (met_help@ucar.edu).

The details of these programs are described more in the chapters in this user's guide.

Chapter 2: Software Installation

Table of Contents

Introduction

The WRF modeling system software installation is fairly straightforward on the ported platforms listed below. The model-component portion of the package is mostly self-contained, meaning that WRF model requires no external libraries (such as for FFTs or various linear algebra solvers). Contained within the WRF system is the WRF-Var component, which has several external libraries that the user must install (for various observation types, FFTs, and linear algebra solvers). Similarly, the WPS package, separate from the WRF source code, has additional external libraries that must be built (in support of Grib2 processing). The one external package that all of the systems require is the netCDF library, which is one of the supported I/O API packages. The netCDF libraries or source code are available from the Unidata homepage at http://www.unidata.ucar.edu (select DOWNLOADS, registration required).

There are three tar files for the WRF code. The first is the WRF model (including the real and ideal pre-processors). The second is the WRF-Var code. This separate tar file must be combined with the WRF code for the WRF-Var code to work. The third tar file is for WRF chemistry. Again, in order to run the WRF chemistry code, both the WRF model and the chemistry tar file must be combined.

The WRF model has been successfully ported to a number of Unix-based machines. We do not have access to all of them and must rely on outside users and vendors to supply the required configuration information for the compiler and loader options. Below is a list of the supported combinations of hardware and software for WRF.

Vendor	Hardware	OS	Compiler
Cray	X1	UniCOS	vendor

Cray	AMD	Linux	PGI / PathScale
IBM	Power Series	AIX	vendor
SGI	IA64 / Opteron	Linux	Intel
COTS*	IA32	Linux	Intel / PGI / gfortran / g95 / PathScale
COTS	IA64 / Opteron	Linux	Intel / PGI / gfortran / PathScale
Mac	Power Series	Darwin	xlf / g95 / PGI / Intel
Mac	Intel	Darwin	g95 / PGI / Intel

* Commercial Off The Shelf systems

The WRF model may be built to run on a single processor machine, a shared-memory machine (that use the OpenMP API), a distributed memory machine (with the appropriate MPI libraries), or on a distributed cluster (utilizing both OpenMP and MPI). The WRF-Var and WPS packages run on the above listed systems.

Required Compilers and Scripting Languages

The WRF model, WPS, and WRF-Var are written in Fortran (what many refer to as Fortran 90). The software layer, RSL_LITE, which sits between WRF and WRF-Var and the MPI interface is written in C. WPS makes direct calls to the MPI libraries for distributed memory message passing. There are also ancillary programs that are written in C to perform file parsing and file construction, which are required for default building of the WRF modeling code. Additionally, the WRF build mechanism uses several scripting languages: including perl, Cshell and Bourne shell. The traditional UNIX text/file processing utilities are used: make, m4, sed, and awk. See Chapter 7: WRF Software (Required Software) for a more detailed listing of the necessary pieces for the WRF build.

Required/Optional Libraries to Download

The only library that is *almost always* required is the netCDF package from Unidata (login > Downloads > NetCDF). Most of the WRF post-processing packages assume that the data from the WRF model, the WPS package, or the WRF-Var program are using the

netCDF libraries. One may also need to add /path-to-netcdf/netcdf/bin to your path so that one may execute netcdf commands, such as **ncdump**.

Hint: If one wants to compile WRF system components on a Linux system that has access to multiple compilers, link the correct external libraries. For example, do not link the libraries built with PathScale when compiling the WRF components with gfortran.

If you are going to be running distributed memory WRF jobs, you need a version of MPI. You can pick up a version of mpich, but you might want your system group to install the code. A working installation of MPI is required prior to a build of WRF using distributed memory. Either MPI-1 or MPI-2 are acceptable. Do you already have an MPI lying around? Try

```
which mpif90
which mpicc
which mpirun
```

If these are all defined executables, you are probably OK. Make sure your paths are set up to point to the MPI **`lib`**, **`include`**, and **`bin`** directories.

Note that to output WRF model data in Grib1 format, Todd Hutchinson (WSI) has provided a complete source library that is included with the software release. However, when trying to link the WPS, the WRF model, and the WRF-Var data streams together, always use the netCDF format.

Post-Processing Utilities

The more widely used (and therefore supported) WRF post-processing utilities are:

- NCL (homepage and WRF download)
  - NCAR Command Language written by NCAR Scientific Computing Division
  - NCL scripts written and maintained by WRF support
  - many template scripts are provided that are tailored for specific real-data and ideal-data cases
  - raw WRF output can be input with the NCL scripts
  - interactive or command-file driven
- Vis5D (homepage and WRF download)
  - download Vis5D executable, build format converter
  - programs are available to convert the WRF output into an input format suitable for Vis5D
  - GUI interface, 3D movie loops, transparency
- GrADS (homepage and WRF download)
  - download GrADS executable, build format converter
  - programs are available to convert the WRF output into an input format suitable for GrADS

    - interpolates to regular lat/lon grid
    - simple to generate publication quality
- RIP (homepage and WRF download)
    - RIP4 written and maintained by Mark Stoelinga, UW
    - interpolation to various surfaces, trajectories, hundreds of diagnostic calculations
    - Fortran source provided
    - based on the NCAR Graphics package
    - pre-processor converts WRF, WPS, and WRF-Var data to RIP input format
    - table driven

UNIX Environment Settings

There are only a few environmental settings that are WRF system related. Most of these are not required, but when things start acting badly, test some out. In Cshell syntax:

- **`setenv WRF_EM_CORE 1`**
    - explicitly defines which model core to build
- **`setenv NETCDF /usr/local/netcdf`** (or where ever you have it stuck)
    - all of the WRF components want both the lib and the include directories

- **`setenv OMP_NUM_THREADS`** ***n*** (where ***n*** is the number of procs to use)
    - if you have OpenMP on your system, this is how to specify the number of threads
- **`setenv MP_STACK_SIZE 64000000`**
    - OpenMP blows through the stack size, set it large
- **`unlimit`**
    - especially if you are on a small system

Building the WRF Code

The WRF code has a fairly complicated build mechanism. It tries to determine the architecture that you are on, and then presents you with options to allow you to select the preferred build method. For example, if you are on a Linux machine, it determines whether this is a 32 or 64 bit machine, and then prompts you for the desired usage of processors (such as serial, shared memory, or distributed memory). You select from among the available compiling options in the build mechanism. For example, do not choose a PGI build if you do not have PGI compilers installed on your system.

- Get the WRF zipped tar file
    - WRFV3 from **`http://www.mmm.ucar.edu/wrf/users/get_source.html`**
    - always get the latest version if you are not trying to continue a long project
- unzip and untar the file
    - **`gzip -cd WRFV3.0.TAR.gz | tar -xf -`**

- **`cd WRFV3`**
- **`./configure`**
  - **serial** means single processor
  - **smpar** means Shared Memory Parallel (OpenMP)
  - **dmpar** means Distributed Memory Parallel (MPI)
  - **dm+sm** means Distributed Memory with Shared Memory (for example, MPI across nodes with OpenMP within a node)
  - the second option is for nesting: 0 = no nesting, 1 = standard nesting, 2 = nesting with a prescribed set of moves, 3 = nesting that allows a domain to follow a vortex (typhoon tracking)
- **`./compile em_real`** (or any of the directory names in ./WRFV3/test)
- **`ls -ls main/*.exe`**
  - if you built a real-data case, you should see **ndown.exe**, **real.exe**, and **wrf.exe**
  - if you built an ideal-data case, you should see **ideal.exe** and **wrf.exe**

Users wishing to run the WRF-Var code or the WRF chemistry code must first download the WRF model tar file, and untar it. Then the WRF-Var (or chemistry) code is untar'ed in the WRFV3 directory (there already exists the appropriate directories). Once the source code from the tar files is combined, then users may proceed with the WRF-Var or WRF chemistry build.

Building the WPS Code

Building WPS requires that WRFV3 is already built.

- Get the WPS zipped tar file
  - WPS.TAR.gz from **`http://www.mmm.ucar.edu/wrf/users/get_source.html`**
- unzip and untar the file
  - **`gzip -cd WPS.TAR.gz | tar -xf -`**
- **`cd WPS`**
- **`./configure`**
  - choose one of the options
  - usually, option "1" and option "2" are for serial builds, that is the best for an initial test
  - WPS requires that you build for the appropriate Grib decoding, select an option that suitable for the data you will use with the ungrib program
- **`./compile`**
- **`ls -ls *.exe`**
  - you should see **`geogrid.exe`**, **`ungrib.exe`**, and **`metgrid.exe`**
- **`ls -ls util/*.exe`**
  - you should see a number of utility executables: **`avg_tsfc.exe`**, **`g1print.exe`**, **`g2print.exe`**, **`mod_levs.exe`**, **`plotfmt.exe`**, **`plotgrids.exe`**, and **`rd_intermediate.exe`**

Building the WRF-Var Code

WRF-Var uses the same build mechanism as WRF, and as a consequence, this mechanism must be instructed to configure and build the code for WRF-Var rather than WRF. Additionally, the paths to libraries needed by WRF-Var code must be set, as described in the steps below.

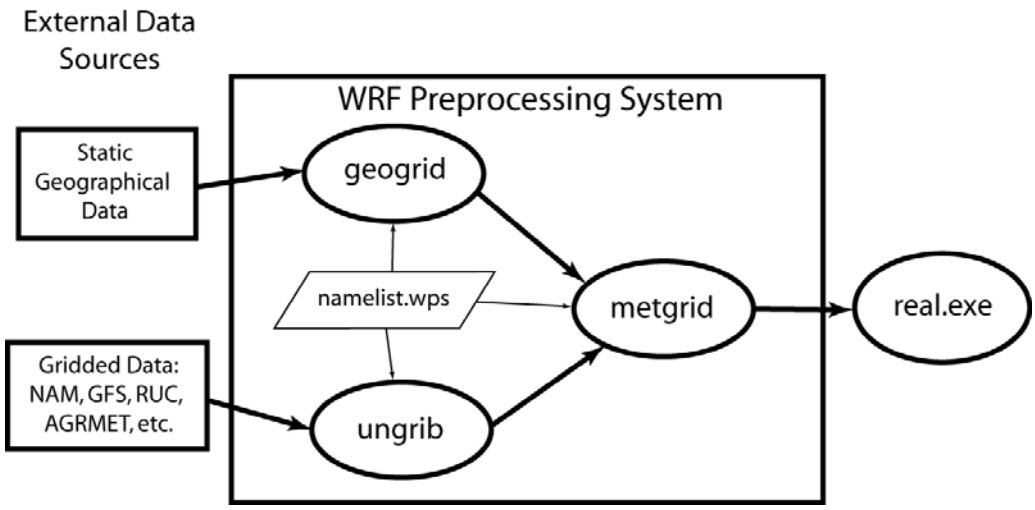
- Get the WRF-Var zipped tar file from **`http://www.mmm.ucar.edu/wrf/users/get_source.html`**
  - While WRF-Var uses the same framework as the WRF model, the code is in a separate tar file. First, the WRF model tar file is downloaded and untar'ed (described in Building the WRF Code).
- Unzip and untar the file WRF-Var code in the WRFV3 directory
  - **`cd WRFV3`**
  - **`gzip -cd WRFVARV3.0.TAR.gz | tar -xf -`**
- **`setenv WRF_DA_CORE 1`**
  - By setting WRF_DA_CORE (where DA refers to "data assimilation"), the build mechanism is instructed to configure and compile WRF-Var rather than WRF.
- Set up environment variables pointing to additional libraries required by WRF-Var, namely, BLAS, LAPACK, and BUFR. These libraries must be installed separately, and can be freely downloaded from **`http://netlib.org/blas/`**, **`http://netlib.org/lapack/`**, and **`http://www.nco.ncep.noaa.gov/sib/decoders/BUFRLIB/`**. Assuming, for example, that these libraries have been installed in subdirectories of /usr/local, the necessary environment variables might be set with
  - **`setenv BLAS /usr/local/blas`**
  - **`setenv LAPACK /usr/local/lapack`**
  - **`setenv BUFR /usr/local/bufr`**
- **`./configure`**
  - **serial** means single processor
  - **smpar** means Shared Memory Parallel (OpenMP)
  - **dmpar** means Distributed Memory Parallel (MPI)
  - **dm+sm** means Distributed Memory with Shared Memory (for example, MPI across nodes with OpenMP within a node)
  - the second option is for nesting; since WRF-Var does not work with nests, option 1 can be selected.
- **`./compile all_wrfvar`**
- **`ls -L var/da/*.exe`**
  - If the compilation was successful, da_wrfvar.exe, da_update_bc.exe, and other executables should be found in the var/da directory.

Chapter 3: WRF Preprocessing System (WPS)

Table of Contents

Introduction

The WRF Preprocessing System (WPS) is a set of three programs whose collective role is to prepare input to the *real* program for real-data simulations. Each of the programs performs one stage of the preparation: *geogrid* defines model domains and interpolates static geographical data to the grids; *ungrib* extracts meteorological fields from GRIB-formatted files; and *metgrid* horizontally interpolates the meteorological fields extracted by ungrib to the model grids defined by geogrid. The work of vertically interpolating meteorological fields to WRF eta levels is now performed within the *real* program, a task that was performed by the *vinterp* program in the WRF SI.



The data flow between the programs of the WPS is shown in the figure above. Each of the WPS programs reads parameters from a common namelist file, as shown in the figure. This namelist file has separate namelist records for each of the programs and a shared namelist record, which defines parameters that are used by more than one WPS program. Not shown in the figure are additional table files that are used by individual programs. These tables provide additional control over the programs' operation, though they generally do not need to be changed by the user. The GEOGRID.TBL and METGRID.TBL files are explained later in this document, though for now, the user need not be concerned with them.

The build mechanism for the WPS, which is very similar to the build mechanism used by the WRF model, provides options for compiling the WPS on a variety of platforms. When MPICH libraries and suitable compilers are available, the metgrid and geogrid programs may be compiled for distributed memory execution, which allows large model domains to be processed in less time. The work performed by the ungrib program is not amenable to parallelization, so ungrib may only be run on a single processor.

Function of Each WPS Program

The WPS consists of three independent programs: *geogrid*, *ungrib*, and *metgrid*. Also included in the WPS are several utility programs, which are described in the section on utility programs. A brief description of each of the three main programs is given below, with further details presented in subsequent sections.

Program geogrid

The purpose of geogrid is to define the simulation domains, and interpolate various terrestrial data sets to the model grids. The simulation domains are defined using

information specified by the user in the “geogrid” namelist record of the WPS namelist file, namelist.wps. By default – and in addition to computing the latitude, longitude, and map scale factors at every grid point – geogrid will interpolate soil categories, land use category, terrain height, annual mean deep soil temperature, monthly vegetation fraction, monthly albedo, maximum snow albedo, and slope category to the model grids. Global data sets for each of these fields are provided through the WRF download page, and only need to be downloaded once. Several of the data sets are available in only one resolution, but others are made available in resolutions of 30”, 2’, 5’, and 10’; here, " denotes arc seconds and ' denotes arc minutes. The user need not download all available resolutions for a data set, although the interpolated fields will generally be more representative if a resolution of data near to that of the simulation domain is used. However, users who expect to work with domains having grid spacings that cover a large range may wish to eventually download all available resolutions of the terrestrial data.

Besides interpolating the default terrestrial fields, the geogrid program is general enough to be able to interpolate most continuous and categorical fields to the simulation domains. New or additional data sets may be interpolated to the simulation domain through the use of the table file, GEOGRID.TBL. The GEOGRID.TBL file defines each of the fields that will be produced by geogrid; it describes the interpolation methods to be used for a field, as well as the location on the file system where the data set for that field is located.

Output from geogrid is written in the WRF I/O API format, and thus, by selecting the NetCDF I/O format, geogrid can be made to write its output in NetCDF for easy visualization using external software packages, including ncview and the new release of RIP4.

Program ungrib

The ungrib program reads GRIB files, "degribs" the data, and writes the data in a simple format, called the intermediate format (see the section on writing data to the intermediate format for details of the format). The GRIB files contain time-varying meteorological fields and are typically from another regional or global model, such as NCEP's NAM or GFS models. The ungrib program can read GRIB Edition 1 and GRIB Edition 2 files.

GRIB files typically contain more fields than are needed to initialize WRF. Both versions of the GRIB format use various codes to identify the variables and levels in the GRIB file. Ungrib uses tables of these codes – called Vtables, for "variable tables" – to define which fields to extract from the GRIB file and write to the intermediate format. Details about the codes can be found in the WMO GRIB documentation and in documentation from the originating center. Vtables for common GRIB model output files are provided with the ungrib software.

Vtables are provided for NAM 104 and 212 grids, the NAM AWIP format, GFS, the NCEP/NCAR Reanalysis archived at NCAR, RUC (pressure level data and hybrid coordinate data), AFWA's AGRMET land surface model output, ECMWF, and other data sets. Users can create their own Vtable for other model output using any of the Vtables as

a template; further details on the meaning of fields in a Vtable are provided in the section on creating and editing Vtables.

Ungrib can write intermediate data files in any one of three user-selectable formats: WPS – a new format containing additional information useful for the downstream programs; SI – the previous intermediate format of the WRF system; and MM5 format, which is included here so that ungrib can be used to provide GRIB2 input to the MM5 modeling system. Any of these formats may be used by WPS to initialize WRF, although the WPS format is recommended.

Program metgrid

The metgrid program horizontally interpolates the intermediate-format meteorological data that are extracted by the ungrib program onto the simulation domains defined by the geogrid program. The interpolated metgrid output can then be ingested by the WRF real program. The range of dates that will be interpolated by metgrid are defined in the "share" namelist record of the WPS namelist file, and date ranges must be specified individually in the namelist for each simulation domain. Since the work of the metgrid program, like that of the ungrib program, is time-dependent, metgrid is run every time a new simulation is initialized.

Control over how each meteorological field is interpolated is provided by the METGRID.TBL file. The METGRID.TBL file provides one section for each field, and within a section, it is possible to specify options such as the interpolation methods to be used for the field, the field that acts as the mask for masked interpolations, and the grid staggering (e.g., U, V in ARW; H, V in NMM) to which a field is interpolated.

Output from metgrid is written in the WRF I/O API format, and thus, by selecting the NetCDF I/O format, metgrid can be made to write its output in NetCDF for easy visualization using external software packages, including the new version of RIP4.

Installing the WPS

The WRF Preprocessing System uses a build mechanism similar to that used by the WRF model. External libraries for geogrid and metgrid are limited to those required by the WRF model, since the WPS uses the WRF model's implementations of the WRF I/O API; consequently, *WRF must be compiled prior to installation of the WPS* so that the I/O API libraries in the WRF external directory will be available to WPS programs. Additionally, the ungrib program requires three compression libraries for GRIB Edition 2 support; however, if support for GRIB2 data is not needed, ungrib can be compiled without these compression libraries.

Required Libraries

The only library that is required to build the WRF model is NetCDF. The user can find the source code, precompiled binaries, and documentation at the UNIDATA home page (http://www.unidata.ucar.edu/software/netcdf/). Most users will select the NetCDF I/O option for WPS due to the easy access to utility programs that support the NetCDF data format, and before configuring the WPS, users should ensure that the environment variable NETCDF is set to the path of the NetCDF installation.

Where WRF adds a software layer between the model and the communications package, the WPS programs geogrid and metgrid make MPI calls directly. Most multi-processor machines come preconfigured with a version of MPI, so it is unlikely that users will need to install this package by themselves.

Three libraries are required by the ungrib program for GRIB Edition 2 compression support. Users are encouraged to engage their system administrators for the installation of these packages so that traditional library paths and include paths are maintained. Paths to user-installed compression libraries are handled in the `configure.wps` file by the `COMPRESSION_LIBS` and `COMPRESSION_INC` variables.

1) JasPer (an implementation of the JPEG2000 standard for "lossy" compression)
http://www.ece.uvic.ca/~mdadams/jasper/
Go down to “JasPer software”, one of the "click here" parts is the source.

```
> ./configure
> make
> make install
```

Note: The GRIB2 libraries expect to find include files in "jasper/jasper.h", so it may be necessary to manually create a "jasper" subdirectory in the "include" directory created by the JasPer installation, and manually link header files there.

2) PNG (compression library for "lossless" compression)
http://www.libpng.org/pub/png/libpng.html
Scroll down to "Source code" and choose a mirror site.

```
> ./configure
> make check
> make install
```

3) zlib (a compression library used by the PNG library)
http://www.zlib.net/
Go to "The current release is publicly available here" section and download.

```
> ./configure
> make
> make install
```

To get around portability issues, the NCEP GRIB libraries, w3 and g2, have been included in the WPS distribution. The original versions of these libraries are available for download from NCEP at http://www.nco.ncep.noaa.gov/pmb/codes/GRIB2/. The specific tar files to download are g2lib and w3lib. Because the ungrib program requires modules from these files, they are not suitable for usage with a traditional library option during the link stage of the build.

Required Compilers and Scripting Languages

The WPS requires the same Fortran and C compilers as were used to build the WRF model, since the WPS executables link to WRF's I/O API libraries. After executing the `./configure` command in the WPS directory, a list of supported compilers on the current system architecture are presented.

WPS Installation Steps

- Download the `WPS.TAR.gz` file and unpack it at the same directory level as WRFV3, as shown below.

```
> ls
-rw-r--r--  1  563863 WPS.TAR.gz
drwxr-xr-x 18    4096 WRFV3

> gzip -d WPS.TAR.gz

> tar xf WPS.TAR

> ls
drwxr-xr-x  7     4096 WPS
-rw-r--r--  1  3491840 WPS.TAR
drwxr-xr-x 18     4096 WRFV3
```

- At this point, a listing of the current working directory should at least include the directories WRFV3 and WPS. First, compile WRF (see the instructions for installing WRF). Then, after the WRF executables are generated, change to the WPS directory and issue the configure command followed by the compile command as below.

```
> cd WPS

> ./configure
```

  - Choose one of the configure options

```
> ./compile >& compile.output
```

- After issuing the compile command, a listing of the current working directory should reveal symbolic links to executables for each of the three WPS programs: geogrid.exe, ungrib.exe, and metgrid.exe. If any of these links do not exist, check the compilation output in compile.output to see what went wrong.

```
> ls
drwxr-xr-x 2   4096 arch
-rwxr-xr-x 1   1672 clean
-rwxr-xr-x 1   3510 compile
-rw-r--r-- 1  85973 compile.output
-rwxr-xr-x 1   4257 configure
-rw-r--r-- 1   2486 configure.wps
drwxr-xr-x 4   4096 geogrid
lrwxrwxrwx 1     23 geogrid.exe -> geogrid/src/geogrid.exe
-rwxr-xr-x 1   1328 link_grib.csh
drwxr-xr-x 3   4096 metgrid
lrwxrwxrwx 1     23 metgrid.exe -> metgrid/src/metgrid.exe
-rw-r--r-- 1   1101 namelist.wps
-rw-r--r-- 1   1987 namelist.wps.all_options
-rw-r--r-- 1   1075 namelist.wps.global
-rw-r--r-- 1    652 namelist.wps.nmm
-rw-r--r-- 1   4786 README
drwxr-xr-x 4   4096 ungrib
lrwxrwxrwx 1     21 ungrib.exe -> ungrib/src/ungrib.exe
drwxr-xr-x 3   4096 util
```

Running the WPS

There are essentially three main steps to running the WRF Preprocessing System:

1. Define a model coarse domain and any nested domains with *geogrid*.
2. Extract meteorological fields from GRIB data sets for the simulation period with *ungrib*.
3. Horizontally interpolate meteorological fields to the model domains with *metgrid*.

When multiple simulations are to be run for the same model domains, it is only necessary to perform the first step once; thereafter, only time-varying data need to be processed for each simulation using steps two and three. Similarly, if several model domains are being run for the same time period using the same meteorological data source, it is not necessary to run ungrib separately for each simulation. Below, the details of each of the three steps are explained.

Step 1: Define model domains with geogrid

In the root of the WPS directory structure, symbolic links to the programs geogrid.exe, ungrib.exe, and metgrid.exe should exist if the WPS software was successfully installed. In addition to these three links, a namelist.wps file should exist. Thus, a listing in the WPS root directory should look something like:

```
> ls
drwxr-xr-x 2   4096 arch
-rwxr-xr-x 1   1672 clean
-rwxr-xr-x 1   3510 compile
-rw-r--r-- 1  85973 compile.output
-rwxr-xr-x 1   4257 configure
-rw-r--r-- 1   2486 configure.wps
```

```
drwxr-xr-x 4   4096 geogrid
lrwxrwxrwx 1     23 geogrid.exe -> geogrid/src/geogrid.exe
-rwxr-xr-x 1   1328 link_grib.csh
drwxr-xr-x 3   4096 metgrid
lrwxrwxrwx 1     23 metgrid.exe -> metgrid/src/metgrid.exe
-rw-r--r-- 1   1101 namelist.wps
-rw-r--r-- 1   1987 namelist.wps.all_options
-rw-r--r-- 1   1075 namelist.wps.global
-rw-r--r-- 1    652 namelist.wps.nmm
-rw-r--r-- 1   4786 README
drwxr-xr-x 4   4096 ungrib
lrwxrwxrwx 1     21 ungrib.exe -> ungrib/src/ungrib.exe
drwxr-xr-x 3   4096 util
```

The model coarse domain and any nested domains are defined in the “geogrid” namelist record of the namelist.wps file, and, additionally, parameters in the “share” namelist record need to be set. An example of these two namelist records is given below, and the user is referred to the description of namelist variables for more information on the purpose and possible values of each variable.

```
&share
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&geogrid
 parent_id         =   1,   1,
 parent_grid_ratio =   1,   3,
 i_parent_start    =   1,  31,
 j_parent_start    =   1,  17,
 s_we              =   1,   1,
 e_we              =  74, 112,
 s_sn              =   1,   1,
 e_sn              =  61,  97,
 geog_data_res     = '10m','2m',
 dx = 30000,
 dy = 30000,
 map_proj = 'lambert',
 ref_lat   = 34.83,
 ref_lon   = -81.03,
 truelat1  =  30.0,
 truelat2  =  60.0,
 stand_lon = -98.,
 geog_data_path = '/mmm/users/wrfhelp/WPS_GEOG/'
/
```

To summarize a set of typical changes to the “share” namelist record relevant to geogrid, the WRF dynamical core must first be selected with `wrf_core`. If WPS is being run for an ARW simulation, `wrf_core` should be set to `'ARW'`, and if running for an NMM simulation, it should be set to `'NMM'`. After selecting the dynamical core, the total number of domains (in the case of ARW) or nesting levels (in the case of NMM) must be chosen with `max_dom`. Since geogrid produces only time-independent data, the `start_date`, `end_date`, and `interval_seconds` variables are ignored by geogrid. Optionally, a location (if not the default, which is the current working directory) where domain files

should be written to may be indicated with the `opt_output_from_geogrid_path` variable, and the format of these domain files may be changed with `io_form_geogrid`.

In the "geogrid" namelist record, the projection of the simulation domain is defined, as are the size and location of all model grids. The map projection to be used for the model domains is specified with the `map_proj` variable, and the namelist variables used to set the parameters of the projection are summarized in the table below.

Map projection / value of `map_proj`	Projection parameters

`'lambert'`	`truelat1`
	`truelat2` (optional)
	`stand_lon`

`'mercator'`	`truelat1`
`'polar'`	`truelat1` `stand_lon`

`'lat-lon'`	`pole_lat`
	`pole_lon`
	`stand_lon`

If WRF is to be run for a regional domain configuration, the location of the coarse domain is determined using the `ref_lat` and `ref_lon` variables, which specify the latitude and longitude, respectively, of the center of the coarse domain. If nested domains are to be processed, their locations with respect to the parent domain are specified with the `i_parent_start` and `j_parent_start` variables; further details of setting up nested domains are provided in the section on nested domains. Next, the dimensions of the coarse domain are determined by the variables `dx` and `dy`, which specify the nominal grid distance in the x-direction and y-direction, and `e_we` and `e_sn`, which give the number of velocity points (i.e., u-staggered or v-staggered points) in the x- and y-directions; for the `'lambert'`, `'mercator'`, and `'polar'` projections, `dx` and `dy` are given in meters, and for the `'lat-lon'` projection, `dx` and `dy` are given in degrees. For nested domains, only the variables `e_we` and `e_sn` are used to determine the dimensions of the grid, and `dx` and `dy` should not be specified for nests, since their values are determined recursively based on the values of the `parent_grid_ratio` and `parent_id` variables, which specify the ratio of a nest's parent grid distance to the nest's grid distance and the grid number of the nest's parent, respectively.

For global WRF simulations, the coverage of the coarse domain is, of course, global, so `ref_lat` and `ref_lon` do not apply, and `dx` *and* `dy` *should not be specified*, since the nominal grid distance is computed automatically based on the number of grid points. Also, it should be noted that the latitude-longitude (`map_proj = 'lat-lon'`) is the only projection in WRF that can support a global domain.

Besides setting variables related to the projection, location, and coverage of model domains, the path to the static geographical data sets must be correctly specified with the `geog_data_path` variable. Also, the user may select which resolution of static data geogrid will interpolate from using the `geog_data_res` variable, whose value should match one of the resolutions of data in the GEOGRID.TBL. If the full set of static data

are downloaded from the WRF download page, possible resolutions include `'30s'`, `'2m'`, `'5m'`, and `'10m'`, corresponding to 30-arc-second data, 2-, 5-, and 10-arc-second data.

Depending on the value of the `wrf_core` namelist variable, the appropriate GEOGRID.TBL file must be used with geogrid, since the grid staggerings that WPS interpolates to differ between dynamical cores. For the ARW, the GEOGRID.TBL.ARW file should be used, and for the NMM, the GEOGRID.TBL.NMM file should be used. Selection of the appropriate GEOGRID.TBL is accomplished by linking the correct file to GEOGRID.TBL in the geogrid directory (or in the directory specified by `opt_geogrid_tbl_path`, if this variable is set in the namelist).

```
> ls geogrid/GEOGRID.TBL

lrwxrwxrwx 1      15 GEOGRID.TBL -> GEOGRID.TBL.ARW
```

For more details on the meaning and possible values for each variable, the user is referred to a description of the namelist variables.

Having suitably defined the simulation coarse domain and nested domains, the geogrid.exe executable may be run to produce domain files. In the case of ARW domains, the domain files are named `geo_em.d0N.nc`, where `N` is the number of the nest defined in each file. When run for NMM domains, geogrid produces the file `geo_nmm.d01.nc` for the coarse domain, and `geo_nmm_nest.l0N.nc` files for each nesting level `N`. Also, note that the file suffix will vary depending on the `io_form_geogrid` that is selected. To run geogrid, issue the following command:

```
> ./geogrid.exe
```

When geogrid.exe has finished running, the message

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Successful completion of geogrid.        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

should be printed, and a listing of the WPS root directory (or the directory specified by `opt_output_from_geogrid_path`, if not this variable was set) should show the domain files. If not, the geogrid.log file may be consulted in an attempt to determine the possible cause of failure. For more information on checking the output of geogrid, the user is referred to the section on checking WPS output.

```
> ls
drwxr-xr-x 2     4096 arch
-rwxr-xr-x 1     1672 clean
-rwxr-xr-x 1     3510 compile
-rw-r--r-- 1    85973 compile.output
-rwxr-xr-x 1     4257 configure
-rw-r--r-- 1     2486 configure.wps
-rw-r--r-- 1  1957004 geo_em.d01.nc
-rw-r--r-- 1  4745324 geo_em.d02.nc
drwxr-xr-x 4     4096 geogrid
lrwxrwxrwx 1       23 geogrid.exe -> geogrid/src/geogrid.exe
-rw-r--r-- 1    11169 geogrid.log
```

```
-rwxr-xr-x 1     1328 link_grib.csh
drwxr-xr-x 3     4096 metgrid
lrwxrwxrwx 1       23 metgrid.exe -> metgrid/src/metgrid.exe
-rw-r--r-- 1     1094 namelist.wps
-rw-r--r-- 1     1987 namelist.wps.all_options
-rw-r--r-- 1     1075 namelist.wps.global
-rw-r--r-- 1      652 namelist.wps.nmm
-rw-r--r-- 1     4786 README
drwxr-xr-x 4     4096 ungrib
lrwxrwxrwx 1       21 ungrib.exe -> ungrib/src/ungrib.exe
drwxr-xr-x 3     4096 util
```

Step 2: Extracting meteorological fields from GRIB files with ungrib

Having already downloaded meteorological data in GRIB format, the first step in extracting fields to the intermediate format involves editing the "share" and "ungrib" namelist records of the namelist.wps file – the same file that was edited to define the simulation domains. An example of the two namelist records is given below.

```
&share
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&ungrib
 out_format = 'WPS',
 prefix     = 'FILE'
/
```

In the "share" namelist record, the variables that are of relevance to ungrib are the starting and ending times of the coarse domain (`start_date` and `end_date`; alternatively, `start_year`, `start_month`, `start_day`, `start_hour`, `end_year`, `end_month`, `end_day`, and `end_hour`) and the interval between meteorological data files (`interval_seconds`). In the "ungrib" namelist record, the variable `out_format` is used to select the format of the intermediate data to be written by ungrib; the metgrid program can read any of the formats supported by ungrib, and thus, any of `'WPS'`, `'SI'`, and `'MM5'` may be specified for `out_format`, although `'WPS'` is recommended. Also in the "ungrib" namelist, the user may specify a path and prefix for the intermediate files with the `prefix` variable. For example, if `prefix` were set to `'ARGRMET'`, then the intermediate files created by ungrib would be named according to AGRMET:*YYYY-MM-DD_HH*, where *YYYY-MM-DD_HH* is the valid time of the data in the file.

After suitably modifying the namelist.wps file, a Vtable must be supplied, and the GRIB files must be linked (or copied) to the filenames that are expected by ungrib. The WPS is supplied with Vtable files for many sources of meteorological data, and the appropriate Vtable may simply be symbolically linked to the file Vtable, which is the Vtable name

expected by ungrib. For example, if the GRIB data are from the GFS model, this could be accomplished with

```
> ln -s ungrib/Variable_Tables/Vtable.GFS Vtable
```

The ungrib program will try to read GRIB files named GRIBFILE.AAA, GRIBFILE.AAB, …, GRIBFILE.ZZZ. In order to simplify the work of linking the GRIB files to these filenames, a shell script, link_grib.csh, is provided. The link_grib.csh script takes as a command-line argument a list of the GRIB files to be linked. For example, if the GRIB data were downloaded to the directory /data/gfs, the files could be linked with link_grib.csh as follows:

```
> ls /data/gfs
-rw-r--r-- 1  42728372 gfs_080324_12_00
-rw-r--r-- 1  48218303 gfs_080324_12_06

> ./link_grib.csh /data/gfs/gfs*
```

After linking the GRIB files and Vtable, a listing of the WPS directory should look something like the following:

```
> ls
drwxr-xr-x 2     4096 arch
-rwxr-xr-x 1     1672 clean
-rwxr-xr-x 1     3510 compile
-rw-r--r-- 1    85973 compile.output
-rwxr-xr-x 1     4257 configure
-rw-r--r-- 1     2486 configure.wps
-rw-r--r-- 1  1957004 geo_em.d01.nc
-rw-r--r-- 1  4745324 geo_em.d02.nc
drwxr-xr-x 4     4096 geogrid
lrwxrwxrwx 1       23 geogrid.exe -> geogrid/src/geogrid.exe
-rw-r--r-- 1    11169 geogrid.log
lrwxrwxrwx 1       38 GRIBFILE.AAA -> /data/gfs/gfs_080324_12_00
lrwxrwxrwx 1       38 GRIBFILE.AAB -> /data/gfs/gfs_080324_12_06
-rwxr-xr-x 1     1328 link_grib.csh
drwxr-xr-x 3     4096 metgrid
lrwxrwxrwx 1       23 metgrid.exe -> metgrid/src/metgrid.exe
-rw-r--r-- 1     1094 namelist.wps
-rw-r--r-- 1     1987 namelist.wps.all_options
-rw-r--r-- 1     1075 namelist.wps.global
-rw-r--r-- 1      652 namelist.wps.nmm
-rw-r--r-- 1     4786 README
drwxr-xr-x 4     4096 ungrib
lrwxrwxrwx 1       21 ungrib.exe -> ungrib/src/ungrib.exe
drwxr-xr-x 3     4096 util
lrwxrwxrwx 1       33 Vtable -> ungrib/Variable_Tables/Vtable.GFS
```

After editing the namelist.wps file and linking the appropriate Vtable and GRIB files, the ungrib.exe executable may be run to produce files of meteorological data in the intermediate format. Ungrib may be run by simply typing the following:

```
> ./ungrib.exe >& ungrib.output
```

Since the ungrib program may produce a significant volume of output, it is recommended that ungrib output be redirected to a file, as in the command above. If ungrib.exe runs successfully, the message

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Successful completion of ungrib.          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

will be written to the end of the ungrib.output file, and the intermediate files should appear in the current working directory. The intermediate files written by ungrib will have names of the form `FILE:`*`YYYY-MM-DD_HH`* (unless, of course, the `prefix` variable was set to a prefix other than `'FILE'`).

```
> ls
drwxr-xr-x 2       4096 arch
-rwxr-xr-x 1       1672 clean
-rwxr-xr-x 1       3510 compile
-rw-r--r-- 1      85973 compile.output
-rwxr-xr-x 1       4257 configure
-rw-r--r-- 1       2486 configure.wps
-rw-r--r-- 1  154946888 FILE:2008-03-24_12
-rw-r--r-- 1  154946888 FILE:2008-03-24_18
-rw-r--r-- 1    1957004 geo_em.d01.nc
-rw-r--r-- 1    4745324 geo_em.d02.nc
drwxr-xr-x 4       4096 geogrid
lrwxrwxrwx 1         23 geogrid.exe -> geogrid/src/geogrid.exe
-rw-r--r-- 1      11169 geogrid.log
lrwxrwxrwx 1         38 GRIBFILE.AAA -> /data/gfs/gfs_080324_12_00
lrwxrwxrwx 1         38 GRIBFILE.AAB -> /data/gfs/gfs_080324_12_06
-rwxr-xr-x 1       1328 link_grib.csh
drwxr-xr-x 3       4096 metgrid
lrwxrwxrwx 1         23 metgrid.exe -> metgrid/src/metgrid.exe
-rw-r--r-- 1       1094 namelist.wps
-rw-r--r-- 1       1987 namelist.wps.all_options
-rw-r--r-- 1       1075 namelist.wps.global
-rw-r--r-- 1        652 namelist.wps.nmm
-rw-r--r-- 1       4786 README
drwxr-xr-x 4       4096 ungrib
lrwxrwxrwx 1         21 ungrib.exe -> ungrib/src/ungrib.exe
-rw-r--r-- 1       1418 ungrib.log
-rw-r--r-- 1      27787 ungrib.output
drwxr-xr-x 3       4096 util
lrwxrwxrwx 1         33 Vtable ->
ungrib/Variable_Tables/Vtable.GFS
```

Step 3: Horizontally interpolating meteorological data with metgrid

In the final step of running the WPS, meteorological data extracted by ungrib are horizontally interpolated to the simulation grids defined by geogrid. In order to run metgrid, the namelist.wps file must be edited. In particular, the “share” and “metgrid” namelist records are of relevance to the metgrid program. Examples of these records are shown below.

```
&share
 wrf_core = 'ARW',
```

```
 max_dom = 2,
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&metgrid
 fg_name                         = 'FILE',
 io_form_metgrid                 = 2,
/
```

By this point, there is generally no need to change any of the variables in the “share” namelist record, since those variables should have been suitably set in previous steps. If the "share" namelist was not edited while running geogrid and ungrib, however, the WRF dynamical core, number of domains, starting and ending times, interval between meteorological data, and path to the static domain files must be set in the “share” namelist record, as described in the steps to run geogrid and ungrib.

In the “metgrid” namelist record, the path and prefix of the intermediate meteorological data files must be given with `fg_name`, the full path and file names of any intermediate files containing constant fields may be specified with the `constants_name` variable, and the output format for the horizontally interpolated files may be specified with the `io_form_metgrid` variable. Other variables in the “metgrid” namelist record, namely, `opt_output_from_metgrid_path` and `opt_metgrid_tbl_path`, allow the user to specify where interpolated data files should be written by metgrid and where the METGRID.TBL file may be found.

As with geogrid and the GEOGRID.TBL file, a METGRID.TBL file appropriate for the WRF core must be linked in the metgrid directory (or in the directory specified by `opt_metgrid_tbl_path`, if this variable is set).

```
> ls metgrid/METGRID.TBL

lrwxrwxrwx 1      15 METGRID.TBL -> METGRID.TBL.ARW
```

After suitably editing the namelist.wps file and verifying that the correct METGRID.TBL will be used, metgrid may be run by issuing the command

```
> ./metgrid.exe
```

If metgrid successfully ran, the message

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Successful completion of metgrid.          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

will be printed. After successfully running, metgrid output files should appear in the WPS root directory (or in the directory specified by `opt_output_from_metgrid_path`, if this variable was set). These files will be named `met_em.d0N.YYYY-MM-DD_HH:mm:ss.nc` in the case of ARW domains, where `N` is the number of the nest whose data reside in the file,

or `met_nmm.d01.YYYY-MM-DD_HH:mm:ss.nc` in the case of NMM domains. Here, `YYYY-MM-DD_HH:mm:ss` refers to the date of the interpolated data in each file. If these files do not exist for each of the times in the range given in the "share" namelist record, the metgrid.log file may be consulted to help in determining the problem in running metgrid.

```
      > ls
      drwxr-xr-x 2       4096 arch
      -rwxr-xr-x 1       1672 clean
      -rwxr-xr-x 1       3510 compile
      -rw-r--r-- 1      85973 compile.output
      -rwxr-xr-x 1       4257 configure
      -rw-r--r-- 1       2486 configure.wps
      -rw-r--r-- 1  154946888 FILE:2008-03-24_12
      -rw-r--r-- 1  154946888 FILE:2008-03-24_18
      -rw-r--r-- 1    1957004 geo_em.d01.nc
      -rw-r--r-- 1    4745324 geo_em.d02.nc
      drwxr-xr-x 4       4096 geogrid
      lrwxrwxrwx 1         23 geogrid.exe -> geogrid/src/geogrid.exe
      -rw-r--r-- 1      11169 geogrid.log
      lrwxrwxrwx 1         38 GRIBFILE.AAA -> /data/gfs/gfs_080324_12_00
      lrwxrwxrwx 1         38 GRIBFILE.AAB -> /data/gfs/gfs_080324_12_06
      -rwxr-xr-x 1       1328 link_grib.csh
      -rw-r--r-- 1    5217648 met_em.d01.2008-03-24_12:00:00.nc
      -rw-r--r-- 1    5217648 met_em.d01.2008-03-24_18:00:00.nc
      -rw-r--r-- 1   12658200 met_em.d02.2008-03-24_12:00:00.nc
      drwxr-xr-x 3       4096 metgrid
      lrwxrwxrwx 1         23 metgrid.exe -> metgrid/src/metgrid.exe
      -rw-r--r-- 1      65970 metgrid.log
      -rw-r--r-- 1       1094 namelist.wps
      -rw-r--r-- 1       1987 namelist.wps.all_options
      -rw-r--r-- 1       1075 namelist.wps.global
      -rw-r--r-- 1        652 namelist.wps.nmm
      -rw-r--r-- 1       4786 README
      drwxr-xr-x 4       4096 ungrib
      lrwxrwxrwx 1         21 ungrib.exe -> ungrib/src/ungrib.exe
      -rw-r--r-- 1       1418 ungrib.log
      -rw-r--r-- 1      27787 ungrib.output
      drwxr-xr-x 3       4096 util
      lrwxrwxrwx 1         33 Vtable ->
ungrib/Variable_Tables/Vtable.GFS
```

Creating Nested Domains with the WPS

To run the WPS for nested-domain simulations is essentially no more difficult than running for a single-domain case; the difference with nested-domain simulations is that the geogrid and metgrid programs process more than one grid when they are run, rather than a single grid for the simulation. In order to specify the size and location of nests, a number of variables in the namelist.wps file must be given lists of values, one value per nest.

```
&share
 wrf_core = 'ARW',
 max_dom = 2,
```

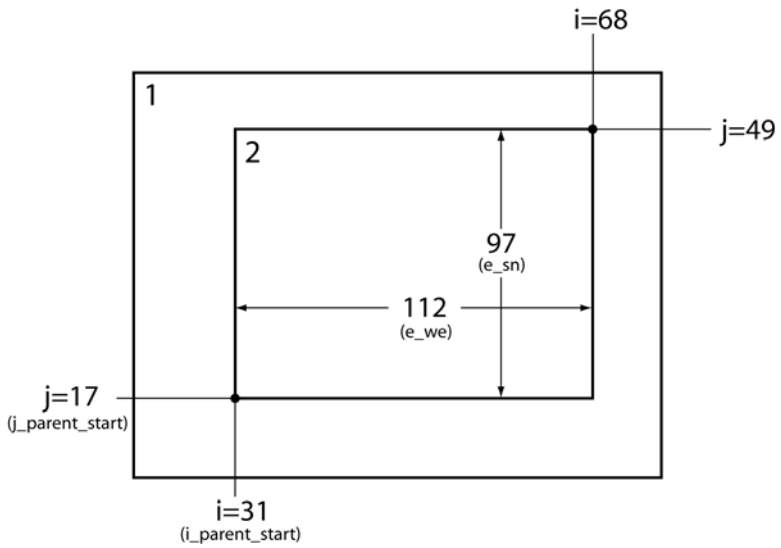
```
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&geogrid
 parent_id         =   1,   1,
 parent_grid_ratio =   1,   3,
 i_parent_start    =   1,  31,
 j_parent_start    =   1,  17,
 s_we              =   1,   1,
 e_we              =  74, 112,
 s_sn              =   1,   1,
 e_sn              =  61,  97,
 geog_data_res     = '10m','2m',
 dx = 30000,
 dy = 30000,
 map_proj = 'lambert',
 ref_lat   = 34.83,
 ref_lon   = -81.03,
 truelat1  =  30.0,
 truelat2  =  60.0,
 stand_lon = -98.
 geog_data_path = '/mmm/users/wrfhelp/WPS_GEOG/'
/
```

The namelist variables that are affected by nests are shown in the (partial) namelist records above. The example shows namelist variables for a two-domain run (the coarse domain plus a single nest), and the effect on the namelist variables generalize to multiple nests in the obvious way: rather than specifying lists of two values, lists of *N* values must be specified, where *N* is the total number of model grids.

In the above example, the first change to the "share" namelist record is to the `max_dom` variable, which must be set to the total number of nests in the simulation, including the coarse domain. Having determined the number of nests, all of the other affected namelist variables must be given a list of *N* values, one for each grid. The only other change to the "share" namelist record is to the starting and ending times. Here, a starting and ending time must be given for each nest, with the restriction that a nest cannot begin before its parent domain or end after its parent domain; also, it is suggested that nests be given starting and ending times that are identical to the desired starting times of the nest *when running WPS*. This is because the nests get their lateral boundary conditions from their parent domain, and thus, only the initial time for a nest needs to be processed by WPS, except when grid nudging, also called analysis nudging, is used in WRF. It is important to note that, *when running WRF*, the actual starting and ending times for all nests must be given in the WRF namelist.input file.

The remaining changes are to the "geogrid" namelist record. In this record, the parent of each nest must be specified with the `parent_id` variable. Every nest must be a child of exactly one other nest, with the coarse domain being its own parent. Related to the identity of a nest's parent is the nest refinement ratio with respect to its parent, which is given by the `parent_grid_ratio` variable; this ratio determines the nominal grid spacing for a nest in relation to the grid spacing of the its parent.



Next, the lower-left corner of a nest is specified as an (*i*, *j*) location in the nest's parent domain; this is done through the `i_parent_start` and `j_parent_start` variables, and the specified location is given with respect to the unstaggered grid. Finally, the dimensions of each nest, in grid points, are given for each nest using the `s_we`, `e_we`, `s_sn`, and `e_sn` variables. The nesting setup in our example namelist is shown in the figure above, where it may be seen how each of the above-mentioned variables is determined. Currently, the starting grid point values in the south-north (`s_sn`) and west-east (`s_we`) directions must be specified as 1, and the ending grid point values (`e_sn` and `e_we`) determine, essentially, the full dimensions of the nest; to ensure that the upper-right corner of the nest's grid is coincident with an unstaggered grid point in the parent domain, both `e_we` and `e_sn` must be one greater than some integer multiple of the nesting ratio. Also, for each nest, the resolution (or list or resolutions; see the description of namelist variables) of source data to interpolate from is specified with the `geog_data_res` variable. For a complete description of these namelist variables, the user is referred to the description of namelist variables.

Using Multiple Meteorological Data Sources

The metgrid program is capable of interpolating time-invariant fields, and it can also interpolate from multiple sources of meteorological data. The first of these capabilities uses the `constants_name` variable in the `&metgrid` namelist record. This variable may be set to a list of filenames – including path information where necessary – of intermediate-formatted files which contains time-invariant fields, and which should be used in the output for every time period processed by metgrid. For example, short simulations may use a constant SST field; this field need only be available at a single time, and may be used by setting the `constants_name` variable to the path and filename of the SST intermediate file. Typical uses of `constants_name` might look like

```
&metgrid
 constants_name = '/data/ungribbed/constants/SST_FILE:2006-08-16_12'
/
```

or

```
&metgrid
 constants_name = 'LANDSEA', 'SOILHGT'
/
```

The second metgrid capability – that of interpolating data from multiple sources – may be useful in situations where two or more complementary data sets need to be combined to produce the full input data needed by real.exe. To interpolate from multiple sources of time-varying, meteorological data, the `fg_name` variable in the `&metgrid` namelist record should be set to a list of prefixes of intermediate files, including path information when necessary. When multiple path-prefixes are given, and the same meteorological field is available from more than one of the sources, data from the last-specified source will take priority over all preceding sources. Thus, data sources may be prioritized by the order in which the sources are given.

As an example of this capability, if surface fields are given in one data source and upper-air data are given in another, the values assigned to the `fg_name` variable may look something like:

```
&metgrid
 fg_name = '/data/ungribbed/SFC', '/data/ungribbed/UPPER_AIR'
/
```

To simplify the process of extracting fields from GRIB files, the `prefix` namelist variable in the `&ungrib` record may be employed. This variable allows the user to control the names of (and paths to) the intermediate files that are created by ungrib. The utility of this namelist variable is most easily illustrated by way of an example. Suppose we wish to work with the North American Regional Reanalysis (NARR) data set, which is split into separate GRIB files for 3-dimensional atmospheric data, surface data, and fixed-field data. We may begin by linking all of the "3D" GRIB files using the `link_grib.csh` script, and by linking the NARR Vtable to the filename `Vtable`. Then, we may suitably edit the `&ungrib` namelist record before running ungrib.exe so that the resulting intermediate files have an appropriate prefix:

```
&ungrib
 out_format = 'WPS',
 prefix = 'NARR_3D',
/
```

After running ungrib.exe, the following files should exist (with a suitable substitution for the appropriate dates):

```
NARR_3D:1979-01-01_00
NARR_3D:1979-01-01_03
NARR_3D:1979-01-01_06
...
```

Given intermediate files for the 3-dimensional fields, we may process the surface fields by linking the surface GRIB files and changing the `prefix` variable in the namelist:

```
&ungrib
 out_format = 'WPS',
 prefix = 'NARR_SFC',
/
```

Again running ungrib.exe, the following should exist in addition to the NARR_3D files:

```
NARR_SFC:1979-01-01_00
NARR_SFC:1979-01-01_03
NARR_SFC:1979-01-01_06
...
```

Finally, the fixed file is linked with the `link_grib.csh` script, and the `prefix` variable in the namelist is again set:

```
&ungrib
 out_format = 'WPS',
 prefix = 'NARR_FIXED',
/
```

Having run ungrib.exe for the third time, the fixed fields should be available in addition to the surface and "3D" fields:

```
NARR_FIXED:1979-11-08_00
```

For the sake of clarity, the fixed file may be renamed to remove any date information, for example, by renaming to simply `NARR_FIXED`, since the fields in the file are static. In this example, we note that the NARR fixed data are only available at a specific time, 1979 November 08 at 0000 UTC, and thus, the user would need to set the correct starting and ending time for the data in the `&share` namelist record before running ungrib on the NARR fixed file; of course, the times should be re-set before metgrid is run.

Given intermediate files for all three parts of the NARR data set, metgrid.exe may be run after the `constants_name` and `fg_name` variables in the `&metgrid` namelist record are set:

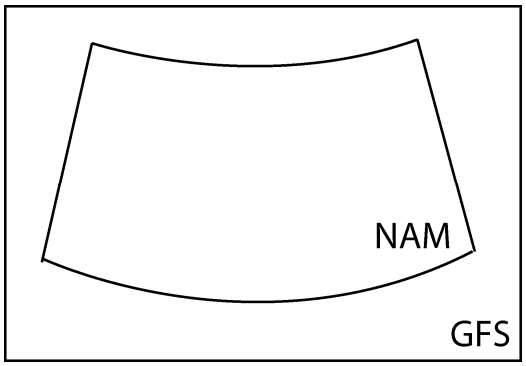
```
&metgrid
 constants_name = 'NARR_FIXED',
 fg_name = 'NARR_3D', 'NARR_SFC'
/
```

Although less common, another situation where multiple data sources would be required is when a source of meteorological data from a regional model is insufficient to cover the entire simulation domain, and data from a larger regional model, or a global model, must be used when interpolating to the remaining points of the simulation grid.

For example, to use NAM data wherever possible, and GFS data elsewhere, the following values might be assigned in the namelist:

```
&metgrid
 fg_name = '/data/ungribbed/GFS', '/data/ungribbed/NAM'
/
```

Then the resulting model domain would use data as shown in the figure below.



If no field is found in more than one source, then no prioritization need be applied by metgrid, and each field will simply be interpolated as usual; of course, each source should cover the entire simulation domain to avoid areas of missing data.

Parallelism in the WPS

If the dimensions of the domains to be processed by the WPS become too large to fit in the memory of a single CPU, it is possible to run the geogrid and metgrid programs in a distributed memory configuration. In order to compile geogrid and metgrid for distributed memory execution, the user must have MPI libraries installed on the target machine, and must have compiled WPS using one of the "DM parallel" configuration options. Upon successful compilation, the geogrid and metgrid programs may be run with the *mpirun* or *mpiexec* commands or through a batch queuing system, depending on the machine.

As mentioned earlier, the work of the ungrib program is not amenable to parallelization, and, further, the memory requirements for ungrib's processing are independent of the memory requirements of geogrid and metgrid; thus, ungrib is always compiled for a single processor and run on a single CPU, regardless of whether a "DM parallel" configuration option was selected during configuration.

Each of the standard WRF I/O API formats (NetCDF, GRIB1, binary) has a corresponding parallel format, whose number is given by adding 100 to the io_form value (e.g., `io_form_geogrid`) for the standard format. It is not necessary to use a parallel io_form, but when one is used, each CPU will read/write its input/output to a separate file, whose name is simply the name that would be used during serial execution, but with

a four-digit processor ID appended to the name. For example, running geogrid on four processors with `io_form_geogrid=102` would create output files named geo_em.d01.nc.0000, geo_em.d01.nc.0001, geo_em.d01.nc.0002, and geo_em.d01.nc.0003 for the coarse domain.

During distributed-memory execution, model domains are decomposed into rectangular patches, with each processor working on a single patch. When reading/writing from/to the WRF I/O API format, each processor reads/writes only its patch. Consequently, if a parallel io_form is chosen for the output of geogrid, metgrid must be run using the same number of processors as were used to run geogrid. Similarly, if a parallel io_form is chosen for the metgrid output files, the real program must be run using the same number of processors. Of course, it is still possible to use a standard io_form when running on multiple processors, in which case all data for the model domain will be distributed/collected upon input/output. As a final note, when geogrid or metgrid are run on multiple processors, each processor will write its own log file, with the log file names being appended with the same four-digit processor ID numbers that are used for the I/O API files.

Checking WPS Output

When running the WPS, it may be helpful to examine the output produced by the programs. For example, when determining the location of nests, it may be helpful to see the interpolated static geographical data and latitude/longitude fields. As another example, when importing a new source of data into WPS – either static data or meteorological data – it can often be helpful to check the resulting interpolated fields in order to make adjustments the interpolation methods used by geogrid or metgrid.

By using the NetCDF format for the geogrid and metgrid I/O forms, a variety of visualization tools that read NetCDF data may be used to check the domain files processed by geogrid or the horizontally interpolated meteorological fields produced by metgrid. In order to set the file format for geogrid and metgrid to NetCDF, the user should specify 2 as the `io_form_geogrid` and `io_form_metgrid` in the WPS namelist file:

```
&share
 io_form_geogrid = 2,
/

&metgrid
 io_form_metgrid = 2,
/
```

Among the available tools, the ncdump, ncview, and new RIP4 programs may be of interest. The ncdump program is a compact utility distributed with the NetCDF libraries that lists the variables and attributes in a NetCDF file. This can be useful, in particular, for checking the domain parameters (e.g., west-east dimension, south-north dimension, or domain center point) in geogrid domain files, or for listing the fields in a file. The ncview program provides an interactive way to view fields in NetCDF files. Also, for users

wishing to produce plots of fields suitable for use in publications, the new release of the RIP4 program may be of interest. The new RIP4 is capable of plotting horizontal contours, map backgrounds, and overlaying multiple fields within the same plot.

Output from the ungrib program is always written in a simple binary format (either 'WPS', 'SI', or 'MM5'), so software for viewing NetCDF files will almost certainly be of no use. However, an NCAR Graphics-based utility, *plotfmt*, is supplied with the WPS source code. This utility produces contour plots of the fields found in an intermediate-format file. If the NCAR Graphics libraries are properly installed, the plotfmt program is automatically compiled, along with other utility programs, when WPS is built.

WPS Utility Programs

Besides the three main WPS programs – geogrid, ungrib, and metgrid – there are a number of utility programs that come with the WPS, and which are compiled in the util directory. These utilities may be used to examine data files, visualize the location of nested domains, compute pressure fields, and compute average surface temperature fields.

A. avg_tsfc.exe

The avg_tsfc.exe program computes a daily mean surface temperature given input files in the intermediate format. Based on the range of dates specified in the "share" namelist section of the namelist.wps file, and also considering the interval between intermediate files, avg_tsfc.exe will use as many complete days' worth of data as possible in computing the average, beginning at the starting date specified in the namelist. If a complete day's worth of data is not available, no output file will be written, and the program will halt as soon as this can be determined. Similarly, any intermediate files for dates that cannot be used as part of a complete 24-hour period are ignored; for example, if there are five intermediate files available at a six-hour interval, the last file would be ignored. The computed average field is written to a new file named TAVGSFC using the same intermediate format version as the input files. This daily mean surface temperature field can then be ingested by metgrid by specifying 'TAVGSFC' for the `constants_name` variable in the "metgrid" namelist section.

B. mod_levs.exe

The mod_levs.exe program is used to remove levels of data from intermediate format files. The levels which are to be kept are specified in new namelist record in the namelist.wps file:

```
&mod_levs
 press_pa = 201300 , 200100 , 100000 ,
             95000 ,  90000 ,
             85000 ,  80000 ,
```

```
              75000 ,   70000 ,
              65000 ,   60000 ,
              55000 ,   50000 ,
              45000 ,   40000 ,
              35000 ,   30000 ,
              25000 ,   20000 ,
              15000 ,   10000 ,
               5000 ,    1000
/
```

Within the `&mod_levs` namelist record, the variable `press_pa` is used to specify a list of levels to keep; the specified levels should match values of `xlvl` in the intermediate format files (see the discussion of the WPS intermediate format for more information on the fields of the intermediate files). The mod_levs program takes two command-line arguments as its input. The first argument is the name of the intermediate file to operate on, and the second argument is the name of the output file to be written.

Removing all but a specified subset of levels from meteorological data sets is particularly useful, for example, when one data set is to be used for the model initial conditions and a second data set is to be used for the lateral boundary conditions. This can be done by providing the initial conditions data set at the first time period to be interpolated by metgrid, and the boundary conditions data set for all other times. If the both data sets have the same number of vertical levels, then no work needs to be done; however, when these two data sets have a different number of levels, it will be necessary, at a minimum, to remove ($m - n$) levels, where $m > n$ and m and n are the number of levels in each of the two data sets, from the data set with m levels. The necessity of having the same number of vertical levels in all files is due to a limitation in real.exe, which requires a constant number of vertical levels to interpolate from.

The mod_levs utility is something of a temporary solution to the problem of accommodating two or more data sets with differing numbers of vertical levels. Should a user choose to use mod_levs, it should be noted that, although the vertical locations of the levels need not match between data sets, all data sets should have a surface level of data, and, when running real.exe and wrf.exe, the value of `p_top` must be chosen to be below the lowest top among the data sets.

C. calc_ecmwf_p.exe

In the course of vertically interpolating meteorological fields, the real program requires a 3d pressure field on the same levels as the other atmospheric fields. The calc_ecmwf_p.exe utility may be used to create such a pressure field for use with ECMWF sigma-level data sets. Given a surface pressure field or (log of surface pressure field) and a list of coefficients A and B, calc_ecmwf_p.exe computes the pressure at an ECMWF sigma level k at grid point (i,j) as $P_{ijk} = A_k + B_k * Psfc_{ij}$. The list of coefficients can be copied from a table appropriate to the number of sigma levels in the data set from http://www.ecmwf.int/products/data/technical/model_levels/index.html. This table should be written in plain text to a file, ecmwf_coeffs, in the current working directory; for example, with 16 sigma levels, the file emcwf_coeffs would contain something like:

```
 0         0.000000      0.000000000
 1      5000.000000      0.000000000
 2      9890.519531      0.001720764
 3     14166.304688      0.013197623
 4     17346.066406      0.042217135
 5     19121.152344      0.093761623
 6     19371.250000      0.169571340
 7     18164.472656      0.268015683
 8     15742.183594      0.384274483
 9     12488.050781      0.510830879
10      8881.824219      0.638268471
11      5437.539063      0.756384850
12      2626.257813      0.855612755
13       783.296631      0.928746223
14         0.000000      0.972985268
15         0.000000      0.992281914
16         0.000000      1.000000000
```

Given a set of intermediate files produced by ungrib and the file ecmwf_coeffs, calc_ecmwf_p loops over all time periods in namelist.wps, and produces an additional intermediate file, PRES:*YYYY-MM-DD_HH*, for each time, which contains pressure data for each full sigma level as well as a 3d relative humidity field. This intermediate file should be specified to metgrid, along with the intermediate data produced by ungrib, by adding 'PRES' to the list of prefixes in the `fg_name` namelist variable.

D. plotgrids.exe

The plotgrids.exe program is an NCAR Graphics-based utility whose purpose is to plot the locations of all nests defined in the namelist.wps file. The program operates on the namelist.wps file, and thus, may be run without having run any of the three main WPS programs. Upon successful completion, plotgrids produces an NCAR Graphics metafile, gmeta, which may be viewed using the idt command. The coarse domain is drawn to fill the plot frame, a map outline with political boundaries is drawn over the coarse domain, and any nested domains are drawn as rectangles outlining the extent of each nest. This utility may be useful particularly during initial placement of domains, at which time the user can iteratively adjust the locations of nests by editing the namelist.wps file, running plotgrids.exe, and determining a set of adjustments to the nest locations. *Currently, this utility does not work for ARW domains that use the latitude-longitude projection* (i.e., when `map_proj = 'lat-lon'`).

E. g1print.exe

The g1print.exe program takes as its only command-line argument the name of a GRIB Edition 1 file. The program prints a listing of the fields, levels, and dates of the data in the file.

F. g2print.exe

Similar to g1print.exe, the g2print.exe program takes as its only command-line argument the name of a GRIB Edition 2 file. The program prints a listing of the fields, levels, and dates of the data in the file.

G. plotfmt.exe

The plotfmt.exe is an NCAR Graphics program that plots the contents of an intermediate format file. The program takes as its only command-line argument the name of the file to plot, and produces an NCAR Graphics metafile, which contains contour plots of each field in input file. The graphics metafile output, gmeta, may be viewed with the idt command, or converted to another format using utilities such as ctrans.

H. rd_intermediate.exe

Given the name of a singe intermediate format file on the command line, the rd_intermediate.exe program prints information about the fields contained in the file.

Writing Meteorological Data to the Intermediate Format

The role of the ungrib program is to decode GRIB data sets into a simple intermediate format that is understood by metgrid. If meteorological data are not available in GRIB Edition 1 or GRIB Edition 2 formats, the user is responsible for writing such data into the intermediate file format. Fortunately, the intermediate format is relatively simple, consisting of a sequence of unformatted Fortran writes. It is important to note that these *unformatted writes use big-endian byte order*, which can typically be specified with compiler flags. Below, we describe the WPS intermediate format; users interested in the SI or MM5 intermediate formats can first gain familiarity with the WPS format, which is very similar, and later examine the Fortran subroutines that read and write all three intermediate formats (metgrid/src/read_met_module.F90 and metgrid/src/write_met_module.F90, respectively).

When writing data to the WPS intermediate format, 2-dimensional fields are written as a rectangular array of real values. 3-dimensional arrays must be split across the vertical dimension into 2-dimensional arrays, which are written independently. It should also be noted that, for global data sets, either a Gaussian or cylindrical equidistant projection must be used, and for regional data sets, either a Mercator, Lambert conformal, polar stereographic, or cylindrical equidistant may be used. The sequence of writes used to write a single 2-dimensional array in the WPS intermediate format is as follows (note that not all of the variables declared below are used for a given projection of the data).

```
integer :: version              ! Format version (must =5 for WPS format)
integer :: nx, ny               ! x- and y-dimensions of 2-d array
integer :: iproj                ! Code for projection of data in array:
                                !       0 = cylindrical equidistant
                                !       1 = Mercator
                                !       3 = Lambert conformal conic
                                !       4 = Gaussian (global only!)
                                !       5 = Polar stereographic
real :: nlats                   ! Number of latitudes north of equator
                                !       (for Gaussian grids)
real :: xfcst                   ! Forecast hour of data
```

```
real :: xlvl                    ! Vertical level of data in 2-d array
real :: startlat, startlon      ! Lat/lon of point in array indicated by
                                !      startloc string
real :: deltalat, deltalon      ! Grid spacing, degrees
real :: dx, dy                  ! Grid spacing, km
real :: xlonc                   ! Standard longitude of projection
real :: truelat1, truelat2      ! True latitudes of projection
real :: earth_radius            ! Earth radius, km
real, dimension(nx,ny) :: slab ! The 2-d array holding the data
logical :: is_wind_grid_rel     ! Flag indicating whether winds are
                                !      relative to source grid (TRUE) or
                                !      relative to earth (FALSE)
character (len=8)  :: startloc ! Which point in array is given by
                                !      startlat/startlon; set either
                                !      to 'SWCORNER' or 'CENTER  '
character (len=9)  :: field     ! Name of the field
character (len=24) :: hdate     ! Valid date for data YYYY:MM:DD_HH:00:00
character (len=25) :: units     ! Units of data
character (len=32) :: map_source  !  Source model / originating center
character (len=46) :: desc      ! Short description of data


!  1) WRITE FORMAT VERSION
write(unit=ounit) version

!  2) WRITE METADATA
! Cylindrical equidistant
if (iproj == 0) then
      write(unit=ounit) hdate, xfcst, map_source, field, &
                        units, desc, xlvl, nx, ny, iproj
      write(unit=ounit) startloc, startlat, startlon, &
                        deltalat, deltalon, earth_radius

! Mercator
else if (iproj == 1) then
      write(unit=ounit) hdate, xfcst, map_source, field, &
                        units, desc, xlvl, nx, ny, iproj
      write(unit=ounit) startloc, startlat, startlon, dx, dy, &
                        truelat1, earth_radius

! Lambert conformal
else if (iproj == 3) then
      write(unit=ounit) hdate, xfcst, map_source, field, &
                        units, desc, xlvl, nx, ny, iproj
      write(unit=ounit) startloc, startlat, startlon, dx, dy, &
                        xlonc, truelat1, truelat2, earth_radius

! Gaussian
else if (iproj == 4) then
      write(unit=ounit) hdate, xfcst, map_source, field, &
                        units, desc, xlvl, nx, ny, iproj
      write(unit=ounit) startloc, startlat, startlon, &
                               nlats, deltalon, earth_radius

! Polar stereographic
else if (iproj == 5) then
      write(unit=ounit) hdate, xfcst, map_source, field, &
                        units, desc, xlvl, nx, ny, iproj
      write(unit=ounit) startloc, startlat, startlon, dx, dy, &
                        xlonc, truelat1, earth_radius

end if
```

```
!  3) WRITE WIND ROTATION FLAG
write(unit=ounit) is_wind_grid_rel

!  4) WRITE 2-D ARRAY OF DATA
write(unit=ounit) slab
```

Creating and Editing Vtables

Although Vtables are provided for many common data sets, it would be impossible for ungrib to anticipate every possible source of meteorological data in GRIB format. When a new source of data is to be processed by ungrib.exe, the user may create a new Vtable either from scratch, or by using an existing Vtable as an example. In either case, a basic knowledge of the meaning and use of the various fields of the Vtable will be helpful.

Each Vtable contains either seven or eleven fields, depending on whether the Vtable is for a GRIB Edition 1 data source or a GRIB Edition 2 data source, respectively. The fields of a Vtable fall into one of three categories: fields that describe how the data are identified within the GRIB file, fields that describe how the data are identified by the ungrib and metgrid programs, and fields specific to GRIB Edition 2. Each variable to be extracted by ungrib.exe will have one or more lines in the Vtable, with multiple lines for data that are split among different level types – for example, a surface level and upper-air levels. The fields that must be specified for a line, or entry, in the Vtable depends on the specifics of the field and level.

The first group of fields, those that describe how the data are identified within the GRIB file, are given under the column headings of the Vtable shown below.

```
GRIB1| Level| From |  To  |
Param| Type |Level1|Level2|
-----+------+------+------+
```

The "GRIB1 Param" field specifies the GRIB code for the meteorological field, which is a number unique to that field within the data set. However, different data sets may use different GRIB codes for the same field – for example, temperature at upper-air levels has GRIB code 11 in GFS data, but GRIB code 130 in ECMWF data. To find the GRIB code for a field, the g1print.exe and g2print.exe utility program may be used.

Given a GRIB code, the "Level Type", "From Level1", and "From Level2" fields are used to specify which levels a field may be found at. As with the "GRIB1 Param" field, the g1print.exe and g2print.exe programs may be used to find values for the level fields. The meanings of the level fields are dependent on the "Level Type" field, and are summarized in the following table.

Level	Level Type	From Level1	To Level2
Upper-air	100	*	(blank)
Surface	1	0	(blank)
Sea-level	102	0	(blank)
Levels at a specified height AGL	105	Height, in meters, of the level above ground	(blank)
Fields given as layers	112	Starting level for the layer	Ending level for the layer

When layer fields (Level Type 112) are specified, the starting and ending points for the layer have units that are dependent on the field itself; appropriate values may be found with the g1print.exe and g2print.exe utility programs.

The second group of fields in a Vtable, those that describe how the data are identified within the metgrid and real programs, fall under the column headings shown below.

```
| metgrid  | metgrid | metgrid                                  |
| Name     |  Units  | Description                              |
+----------+---------+------------------------------------------+
```

The most important of these three fields is the "metgrid Name" field, which determines the variable name that will be assigned to a meteorological field when it is written to the intermediate files by ungrib. This name should also match an entry in the METGRID.TBL file, so that the metgrid program can determine how the field is to be horizontally interpolated. The "metgrid Units" and "metgrid Description" fields specify the units and a short description for the field, respectively; here, it is important to note that if no description is given for a field, then *ungrib will not write that field out to the intermediate files*.

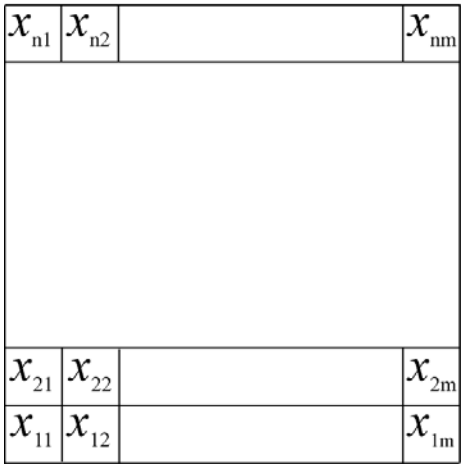
The final group of fields, which provide GRIB2-specific information, are found under the column headings below.

```
|GRIB2|GRIB2|GRIB2|GRIB2|
|Discp|Catgy|Param|Level|
+-----------------------+
```

The GRIB2 fields are only needed in a Vtable that is to be used for GRIB Edition 2 data sets, although having these fields in a Vtable does not prevent that Vtable from also being used for GRIB Edition 1 data. For example, the Vtable.GFS file contains GRIB2 Vtable fields, but is used for both 1-degree (GRIB1) GFS and 0.5-degree (GRIB2) GFS data sets. Since Vtables are provided for most known GRIB Edition 2 data sets, the corresponding Vtable fields are not described here at present.

Writing Static Data to the Geogrid Binary Format

The static geographical data sets that are interpolated by the geogrid program are stored as regular 2-dimensional and 3-dimensional arrays written in a simple binary raster format. Users with a new source for a given static field can ingest their data with WPS by writing the data set into this binary format. The geogrid format is capable of supporting single-level and multi-level continuous fields, categorical fields represented as dominant categories, and categorical fields given as fractional fields for each category. The most simple of these field types in terms of representation in the binary format is a categorical field given as a dominant category at each source grid point, an example of which is the 30-second USGS land use data set.



For a categorical field given as dominant categories, the data must first be stored in a regular 2-dimensional array of integers, with each integer giving the dominant category at the corresponding source grid point. Given this array, the data are written to a file, row-by-row, beginning at the bottom, or southern-most, row. For example, in the figure above, the elements of the $n \times m$ array would be written in the order x_{11}, x_{12}, ..., x_{1m}, x_{21}, ..., x_{2m}, ..., x_{n1}, ..., x_{nm}. When written to the file, every element is stored as a 1-, 2-, 3-, or 4-byte integer in big-endian byte order (i.e., for the 4-byte integer *ABCD*, byte *A* is stored at the lowest address and byte *D* at the highest), although little-endian files may be used by setting `endian=little` in the "index" file for the data set. Every element in a file must use the same number of bytes for its storage, and, of course, it is advantageous to use the fewest number of bytes needed to represent the complete range of values in the array.

Similar in format to a field of dominant categories is the case of a field of continuous, or real, values. Like dominant-category fields, single-level continuous fields are first organized as a regular 2-dimensional array, then written, row-by-row, to a binary file. However, because a continuous field may contain non-integral or negative values, the storage representation of each element within the file is slightly more complex. All elements in the array must first be converted to integral values. This is done by first scaling all elements by a constant, chosen to maintain the required precision, and then

removing any remaining fractional part through rounding. For example, if three decimal places of precision are required, the value -2.71828 would need to be scaled by 1000 and rounded to -2718. Following conversion of all array elements to integral values, if any negative values are found in the array, a second conversion must be applied: if elements are stored using 1 byte each, then 2^8 is added to each negative element; for storage using 2 bytes, 2^{16} is added to each negative element; for storage using 3 bytes, 2^{24} is added to each negative element; and for storage using 4 bytes, a value of 2^{32} is added to each negative element. It is important to note that no conversion is applied to positive elements. Finally, the resulting positive, integral array is written as in the case of a dominant-category field.

Multi-level continuous fields are handled much the same as single-level continuous fields. For an $n \times m \times r$ array, conversion to a positive, integral field is first performed as described above. Then, each $n \times m$ sub-array is written contiguously to the binary file as before, beginning with the smallest r index. Categorical fields that are given as fractional fields for each possible category can be thought of as multi-level continuous fields, where each level k is the fractional field for category k.

When writing a field to a file in the geogrid binary format, the user should adhere to the naming convention used by the geogrid program, which expects data files to have names of the form *xstart-xend.ystart-yend*, where *xstart*, *xend*, *ystart*, and *yend* are five-digit positive integers specifying, respectively, the starting x-index of the array contained in the file, the ending x-index of the array, the starting y-index of the array, and the ending y-index of the array; here, indexing begins at 1, rather than 0. So, for example, an 800 × 1200 array (i.e., 800 rows and 1200 columns) might be named 00001-01200.00001-00800.

When a data set is given in several pieces, each of the pieces may be formed as a regular rectangular array, and each array may be written to a separate file. In this case, the relative locations of the arrays are determined by the range of x- and y-indices in the file names for each of the arrays. It is important to note, however, that every tile must have the same x- and y-dimensions, and that tiles of data within a data set must not overlap; furthermore, all tiles must start and end on multiples of the index ranges. For example, the global 30-second USGS topography data set is divided into arrays of dimension 1200 × 1200, with each array containing a 10-degree × 10-degree piece of the data set; the file whose south-west corner is located at (90S, 180W) is named 00001-01200.00001-01200, and the file whose north-east corner is located at (90N, 180E) is named 42001-43200.20401-21600.

Clearly, since the starting and ending indices must have five digits, a field cannot have more than 99999 data points in either of the x- or y-directions. In case a field has more than 99999 data points in either dimension, the user can simply split the data set into several smaller data sets which will be identified separately to geogrid. For example, a very large global data set may be split into data sets for the Eastern and Western hemispheres.

Besides the binary data files themselves, geogrid requires one extra metadata file per data set. This metadata file is always named 'index', and thus, two data sets cannot reside in the same directory. Essentially, this metadata file is the first file that geogrid looks for when processing a data set, and the contents of the file provide geogrid with all of the information necessary for constructing names of possible data files. The contents of an example index file are given below.

```
type = continuous
signed = yes
projection = regular_ll
dx = 0.00833333
dy = 0.00833333
known_x = 1.0
known_y = 1.0
known_lat = -89.99583
known_lon = -179.99583
wordsize = 2
tile_x = 1200
tile_y = 1200
tile_z = 1
tile_bdr=3
units="meters MSL"
description="Topography height"
```

For a complete listing of keywords that may appear in an index file, along with the meaning of each keyword, the user is referred to the section on index file options.

Description of the Namelist Variables

A. SHARE section

This section provides variables that are used by more than one WPS program. For example, the `wrf_core` variable specifies whether WPS is to produce data for the ARW or the NMM core – information which is needed by both the geogrid and metgrid programs.

1. WRF_CORE : A character string set to either `'ARW'` or `'NMM'` that tells WPS which dynamical core the input data are being prepared for. Default value is `'ARW'`.

2. MAX_DOM : An integer specifying the total number of domains/nests, including the parent domain, in the simulation. Default value is 1.

3. START_YEAR : A list of MAX_DOM 4-digit integers specifying the starting UTC year of the simulation for each nest. No default value.

4. START_MONTH : A list of MAX_DOM 2-digit integers specifying the starting UTC month of the simulation for each nest. No default value.

5. START_DAY : A list of MAX_DOM 2-digit integers specifying the starting UTC day of the simulation for each nest. No default value.

6. START_HOUR : A list of MAX_DOM 2-digit integers specifying the starting UTC hour of the simulation for each nest. No default value.

7. END_YEAR : A list of MAX_DOM 4-digit integers specifying the ending UTC year of the simulation for each nest. No default value.

8. END_MONTH : A list of MAX_DOM 2-digit integers specifying the ending UTC month of the simulation for each nest. No default value.

9. END_DAY : A list of MAX_DOM 2-digit integers specifying the ending UTC day of the simulation for each nest. No default value.

10. END_HOUR : A list of MAX_DOM 2-digit integers specifying the ending UTC hour of the simulation for each nest. No default value.

11. START_DATE : A list of MAX_DOM character strings of the form `'YYYY-MM-DD_HH:mm:ss'` specifying the starting UTC date of the simulation for each nest. The `start_date` variable is an alternate to specifying `start_year`, `start_month`, `start_day`, and `start_hour`, and if both methods are used for specifying the starting time, the `start_date` variable will take precedence. No default value.

12. END_DATE : A list of MAX_DOM character strings of the form `'YYYY-MM-DD_HH:mm:ss'` specifying the ending UTC date of the simulation for each nest. The `end_date` variable is an alternate to specifying `end_year`, `end_month`, `end_day`, and `end_hour`, and if both methods are used for specifying the ending time, the `end_date` variable will take precedence. No default value.

13. INTERVAL_SECONDS : The integer number of seconds between time-varying meteorological input files. No default value.

14. IO_FORM_GEOGRID : The WRF I/O API format that the domain files created by the geogrid program will be written in. Possible options are: 1 for binary; 2 for NetCDF; 3 for GRIB1. When option 1 is given, domain files will have a suffix of .int; when option 2 is given, domain files will have a suffix of .nc; when option 3 is given, domain files will have a suffix of .gr1. Default value is 2 (NetCDF).

15. OPT_OUTPUT_FROM_GEOGRID_PATH : A character string giving the path, either relative or absolute, to the location where output files from geogrid should be written to and read from. Default value is `'./'`.

16. DEBUG_LEVEL : An integer value indicating the extent to which different types of messages should be sent to standard output. When `debug_level` is set to 0, only generally useful messages and warning messages will be written to standard output.

When `debug_level` is greater than 100, informational messages that provide further runtime details are also written to standard output. Debugging messages and messages specifically intended for log files are never written to standard output, but are always written to the log files. Default value is 0.

B. GEOGRID section

This section specifies variables that are specific to the geogrid program. Variables in the geogrid section primarily define the size and location of all model domains, and where the static geographical data are found.

1. PARENT_ID : A list of MAX_DOM integers specifying, for each nest, the domain number of the nest's parent; for the coarsest domain, this variable should be set to 1. Default value is 1.

2. PARENT_GRID_RATIO : A list of MAX_DOM integers specifying, for each nest, the nesting ratio relative to the domain's parent. No default value.

3. I_PARENT_START : A list of MAX_DOM integers specifying, for each nest, the x-coordinate of the lower-left corner of the nest in the parent *unstaggered* grid. For the coarsest domain, a value of 1 should be specified. No default value.

4. J_PARENT_START : A list of MAX_DOM integers specifying, for each nest, the y-coordinate of the lower-left corner of the nest in the parent *unstaggered* grid. For the coarsest domain, a value of 1 should be specified. No default value.

5. S_WE : A list of MAX_DOM integers which should all be set to 1. Default value is 1.

6. E_WE : A list of MAX_DOM integers specifying, for each nest, the nest's full west-east dimension. For nested domains, `e_we` must be one greater than an integer multiple of the nest's `parent_grid_ratio` (i.e., `e_ew` = *n**`parent_grid_ratio`+1 for some positive integer *n*). No default value.

7. S_SN : A list of MAX_DOM integers which should all be set to 1. Default value is 1.

8. E_SN : A list of MAX_DOM integers specifying, for each nest, the nest's full south-north dimension. For nested domains, `e_sn` must be one greater than an integer multiple of the nest's `parent_grid_ratio` (i.e., `e_sn` = *n**`parent_grid_ratio`+1 for some positive integer *n*). No default value.

9. GEOG_DATA_RES : A list of MAX_DOM character strings specifying, for each nest, a corresponding resolution or list of resolutions separated by `+` symbols of source data to be used when interpolating static terrestrial data to the nest's grid. For each nest, this string should contain a resolution matching a string preceding a colon in a `rel_path` or `abs_path` specification (see the description of GEOGRID.TBL options) in the GEOGRID.TBL file for each field. If a resolution in the string does not match any such

string in a rel_path or abs_path specification for a field in GEOGRID.TBL, a default resolution of data for that field, if one is specified, will be used. If multiple resolutions match, the first resolution to match a string in a rel_path or abs_path specification in the GEOGRID.TBL file will be used. Default value is 'default'.

10. DX : A real value specifying the grid distance in the x-direction where the map scale factor is 1. For ARW, the grid distance is in meters for the 'polar', 'lambert', and 'mercator' projection, and in degrees longitude for the 'lat-lon' projection; for NMM, the grid distance is in degrees longitude. Grid distances for nests are determined recursively based on values specified for parent_grid_ratio and parent_id. No default value.

11. DY : A real value specifying the nominal grid distance in the y-direction where the map scale factor is 1. For ARW, the grid distance is in meters for the 'polar', 'lambert', and 'mercator' projection, and in degrees latitude for the 'lat-lon' projection; for NMM, the grid distance is in degrees latitude. Grid distances for nests are determined recursively based on values specified for parent_grid_ratio and parent_id. No default value.

12. MAP_PROJ : A character string specifying the projection of the simulation domain. For ARW, accepted projections are 'lambert', 'polar', 'mercator', and 'lat-lon'; for NMM, a projection of 'rotated_ll' must be specified. Default value is 'lambert'.

13. REF_LAT : A real value specifying the latitude part of a (latitude, longitude) location whose (i,j) location in the simulation domain is known. For ARW, ref_lat gives the latitude of the center-point of the coarse domain by default (i.e., when ref_x and ref_y are not specified). For NMM, ref_lat always gives the latitude to which the origin is rotated. No default value.

14. REF_LON : A real value specifying the longitude part of a (latitude, longitude) location whose (i, j) location in the simulation domain is known. For ARW, ref_lon gives the longitude of the center-point of the coarse domain by default (i.e., when ref_x and ref_y are not specified). For NMM, ref_lon always gives the longitude to which the origin is rotated. For both ARW and NMM, west longitudes are negative, and the value of ref_lon should be in the range [-180, 180]. No default value.

15. REF_X : A real value specifying the i part of an (i, j) location whose (latitude, longitude) location in the simulation domain is known. The (i, j) location is always given with respect to the mass-staggered grid, whose dimensions are one less than the dimensions of the unstaggered grid. Default value is (((E_WE-1.)+1.)/2.) = (E_WE/2.).

16. REF_Y : A real value specifying the j part of an (i, j) location whose (latitude, longitude) location in the simulation domain is known. The (i, j) location is always given with respect to the mass-staggered grid, whose dimensions are one less than the dimensions of the unstaggered grid. Default value is (((E_SN-1.)+1.)/2.) = (E_SN/2.).

17. TRUELAT1 : A real value specifying, for ARW, the first true latitude for the Lambert conformal conic projection, or the only true latitude for the polar stereographic projection. For NMM, `truelat1` is ignored. No default value.

18. TRUELAT2 : A real value specifying, for ARW, the second true latitude for the Lambert conformal conic projection. For NMM, `truelat2` is ignored. No default value.

19. STAND_LON : A real value specifying, for ARW, the longitude that is parallel with the y-axis in conic and azimuthal projections. For NMM, `stand_lon` is ignored. No default value.

20. POLE_LAT : For the latitude-longitude projection for ARW, the latitude of the North Pole with respect to the computational latitude-longitude grid in which -90.0° latitude is at the bottom of a global domain, 90.0° latitude is at the top, and 180.0° longitude is at the center. Default value is 90.0.

21. POLE_LON : For the latitude-longitude projection for ARW, the longitude of the North Pole with respect to the computational lat/lon grid in which -90.0° latitude is at the bottom of a global domain, 90.0° latitude is at the top, and 180.0° longitude is at the center. Default value is 0.0.

22. GEOG_DATA_PATH : A character string giving the path, either relative or absolute, to the directory where the geographical data directories may be found. This path is the one to which `rel_path` specifications in the GEOGRID.TBL file are given in relation to. No default value.

23. OPT_GEOGRID_TBL_PATH : A character string giving the path, either relative or absolute, to the GEOGRID.TBL file. The path should not contain the actual file name, as GEOGRID.TBL is assumed, but should only give the path where this file is located. Default value is `'./geogrid/'`.

C. UNGRIB section

Currently, this section contains only two variables, which determine the output format written by ungrib and the name of the output files.

1. OUT_FORMAT : A character string set either to `'WPS'`, `'SI'`, or `'MM5'`. If set to `'MM5'`, ungrib will write output in the format of the MM5 pregrid program; if set to `'SI'`, ungrib will write output in the format of grib_prep.exe; if set to `'WPS'`, ungrib will write data in the WPS intermediate format. Default value is `'WPS'`.

2. PREFIX : A character string that will be used as the prefix for intermediate-format files created by ungrib; here, prefix refers to the string *PREFIX* in the filename *PREFIX*:*YYYY-MM-DD_HH* of an intermediate file. The prefix may contain path information, either relative or absolute, in which case the intermediate files will be

written in the directory specified. This option may be useful to avoid renaming intermediate files if ungrib is to be run on multiple sources of GRIB data. Default value is `'FILE'`.

D. METGRID section

This section defines variables used only by the metgrid program. Typically, the user will be interested in the `fg_name` variable, and may need to modify other variables of this section less frequently.

1. FG_NAME : A list of character strings specifying the path and prefix of ungribbed data files. The path may be relative or absolute, and the prefix should contain all characters of the filenames up to, but not including, the colon preceding the date. When more than one `fg_name` is specified, and the same field is found in two or more input sources, the data in the last encountered source will take priority over all preceding sources for that field. Default value is an empty list (i.e., no meteorological fields).

2. CONSTANTS_NAME : A list of character strings specifying the path and full filename of ungribbed data files which are time-invariant. The path may be relative or absolute, and the filename should be the complete filename; since the data are assumed to be time-invariant, no date will be appended to the specified filename. Default value is an empty list (i.e., no constant fields).

3. IO_FORM_METGRID : The WRF I/O API format that the output created by the metgrid program will be written in. Possible options are: 1 for binary; 2 for NetCDF; 3 for GRIB1. When option 1 is given, output files will have a suffix of .int; when option 2 is given, output files will have a suffix of .nc; when option 3 is given, output files will have a suffix of .gr1. Default value is 2 (NetCDF).

4. OPT_OUTPUT_FROM_METGRID_PATH : A character string giving the path, either relative or absolute, to the location where output files from metgrid should be written to. The default value is the current working directory (i.e., the default value is `'./'`).

5. OPT_METGRID_TBL_PATH : A character string giving the path, either relative or absolute, to the METGRID.TBL file; the path should not contain the actual file name, as METGRID.TBL is assumed, but should only give the path where this file is located. Default value is `'./metgrid/'`.

6. OPT_IGNORE_DOM_CENTER : A logical value, either `.TRUE.` or `.FALSE.`, specifying whether, for times other than the initial time, interpolation of meteorological fields to points on the interior of the simulation domain should be avoided in order to decrease the runtime of metgrid. This option currently has no effect. Default value is `.FALSE.`.

Description of GEOGRID.TBL Options

The GEOGRID.TBL file is a text file that defines parameters of each of the data sets to be interpolated by geogrid. Each data set is defined in a separate section, with sections being delimited by a line of equality symbols (e.g., '=========='). Within each section, there are specifications, each of which has the form of *keyword=value*. Some keywords are required in each data set section, while others are optional; some keywords are mutually exclusive with other keywords. Below, the possible keywords and their expected range of values are described.

1. NAME : A character string specifying the name that will be assigned to the interpolated field upon output. No default value.

2. PRIORITY : An integer specifying the priority that the data source identified in the table section takes with respect to other sources of data for the same field. If a field has *n* sources of data, then there must be *n* separate table entries for the field, each of which must be given a unique value for `priority` in the range [1,*n*]. No default value.

3. DEST_TYPE : A character string, either `categorical` or `continuous`, that tells whether the interpolated field from the data source given in the table section is to be treated as a continuous or a categorical field. No default value.

4. INTERP_OPTION : A sequence of one or more character strings, which are the names of interpolation methods to be used when horizontally interpolating the field. Available interpolation methods are: `average_4pt`, `average_16pt`, `wt_average_4pt`, `wt_average_16pt`, `nearest_neighbor`, `four_pt`, `sixteen_pt`, `search`, `average_gcell(`*r*`)` ; for the grid cell average method (`average_gcell`), the optional argument *r* specifies the minimum ratio of source data resolution to simulation grid resolution at which the method will be applied; unless specified, *r* = 0.0, and the option is used for any ratio. When a sequence of two or more methods are given, the methods should be separated by a `+` sign. No default value.

5. SMOOTH_OPTION : A character string giving the name of a smoothing method to be applied to the field after interpolation. Available smoothing options are: `1-2-1`, `smth-desmth`, and `smth-desmth_special` (ARW only). Default value is null (i.e., no smoothing is applied).

6. SMOOTH_PASSES : If smoothing is to be performed on the interpolated field, `smooth_passes` specifies an integer number of passes of the smoothing method to apply to the field. Default value is 1.

7. REL_PATH : A character string specifying the path relative to the path given in the namelist variable `geog_data_path`. A specification is of the general form *RES_STRING*:*REL_PATH*, where *RES_STRING* is a character string identifying the source or resolution of the data in some unique way and may be specified in the namelist variable `geog_data_res`, and *REL_PATH* is a path relative to `geog_data_path` where

the index and data tiles for the data source are found. More than one `rel_path` specification may be given in a table section if there are multiple sources or resolutions for the data source, just as multiple resolutions may be specified (in a sequence delimited by `+` symbols) for `geog_data_res`. See also `abs_path`. No default value.

8. ABS_PATH : A character string specifying the absolute path to the index and data tiles for the data source. A specification is of the general form *RES_STRING*:*ABS_PATH*, where *RES_STRING* is a character string identifying the source or resolution of the data in some unique way and may be specified in the namelist variable `geog_data_res`, and *ABS_PATH* is the absolute path to the data source's files. More than one `abs_path` specification may be given in a table section if there are multiple sources or resolutions for the data source, just as multiple resolutions may be specified (in a sequence delimited by `+` symbols) for `geog_data_res`. See also `rel_path`. No default value.

9. OUTPUT_STAGGER : A character string specifying the grid staggering to which the field is to be interpolated. For ARW domains, possible values are `U`, `V`, and `M`; for NMM domains, possible values are `HH` and `VV`. Default value for ARW is `M`; default value for NMM is `HH`.

10. LANDMASK_WATER : An integer value that is the index of the category within the field that represents water. When `landmask_water` is specified in the table section of a field for which `dest_type=categorical`, the LANDMASK field will be computed from the field using the specified category as the water category. The keywords `landmask_water` and `landmask_land` are mutually exclusive. Default value is null (i.e., a landmask will not be computed from the field).

11. LANDMASK_LAND : An integer value that is the index of the category within the field that represents land. When `landmask_water` is specified in the table section of a field for which `dest_type=categorical`, the LANDMASK field will be computed from the field using the specified category as the land category. The keywords `landmask_water` and `landmask_land` are mutually exclusive. Default value is null (i.e., a landmask will not be computed from the field).

12. MASKED : Either `land` or `water`, indicating that the field is not valid at land or water points, respectively. If the `masked` keyword is used for a field, those grid points that are of the masked type (land or water) will be assigned the value specified by `fill_missing`. Default value is null (i.e., the field is not masked).

13. FILL_MISSING : A real value used to fill in any missing or masked grid points in the interpolated field. Default value is 1.E20.

14. HALT_ON_MISSING : Either `yes` or `no`, indicating whether geogrid should halt with a fatal message when a missing value is encountered in the interpolated field. Default value is `no`.

15. DOMINANT_CATEGORY : When specified as a character string, the effect is to cause geogrid to compute the dominant category from the fractional categorical field, and to output the dominant category field with the name specified by the value of `dominant_category`. This option can only be used for fields with `dest_type=categorical`. Default value is null (i.e., no dominant category will be computed from the fractional categorical field).

16. DOMINANT_ONLY : When specified as a character string, the effect is similar to that of the `dominant_category` keyword: geogrid will compute the dominant category from the fractional categorical field and output the dominant category field with the name specified by the value of `dominant_only`. Unlike with `dominant_category`, though, when `dominant_only` is used, the fractional categorical field will not appear in the geogrid output. This option can only be used for fields with `dest_type=categorical`. Default value is null (i.e., no dominant category will be computed from the fractional categorical field).

17. DF_DX : When `df_dx` is assigned a character string value, the effect is to cause geogrid to compute the directional derivative of the field in the x-direction using a central difference along the interior of the domain, or a one-sided difference at the boundary of the domain; the derivative field will be named according to the character string assigned to the keyword `df_dx`. Default value is null (i.e., no derivative field is computed).

18. DF_DY : When `df_dy` is assigned a character string value, the effect is to cause geogrid to compute the directional derivative of the field in the y-direction using a central difference along the interior of the domain, or a one-sided difference at the boundary of the domain; the derivative field will be named according to the character string assigned to the keyword `df_dy`. Default value is null (i.e., no derivative field is computed).

19. Z_DIM_NAME : For 3-dimensional output fields, a character string giving the name of the vertical dimension, or z-dimension. A continuous field may have multiple levels, and thus be a 3-dimensional field, and a categorical field may take the form of a 3-dimensional field if it is written out as fractional fields for each category. No default value.

Description of index Options

Related to the GEOGRID.TBL are the index files that are associated with each static data set. An index file defines parameters specific to that data set, while the GEOGRID.TBL file describes how each of the data sets should be treated by geogrid. As with the GEOGRID.TBL file, specifications in an index file are of the form *keyword=value*. Below are possible keywords and their possible values.

1. PROJECTION : A character string specifying the projection of the data, which may be either `lambert`, `polar`, `mercator`, `regular_ll`, `albers_nad83`, or `polar_wgs84`. No default value.

2. TYPE : A character string, either `categorical` or `continuous`, that determines whether the data in the data files should be interpreted as a continuous field or as discrete indices. For categorical data represented by a fractional field for each possible category, `type` should be set to `continuous`. No default value.

3. SIGNED : Either `yes` or `no`, indicating whether the values in the data files (which are always represented as integers) are signed in two's complement form or not. Default value is `no`.

4. UNITS : A character string, enclosed in quotation marks ("), specifying the units of the interpolated field; the string will be written to the geogrid output files as a variable time-independent attribute. No default value.

5. DESCRIPTION : A character string, enclosed in quotation marks ("), giving a short description of the interpolated field; the string will be written to the geogrid output files as a variable time-independent attribute. No default value.

6. DX : A real value giving the grid spacing in the x-direction of the data set. If `projection` is one of `lambert`, `polar`, `mercator`, `albers_nad83`, or `polar_wgs84`, `dx` gives the grid spacing in meters; if `projection` is `regular_ll`, `dx` gives the grid spacing in degrees. No default value.

7. DY : A real value giving the grid spacing in the y-direction of the data set. If `projection` is one of `lambert`, `polar`, `mercator`, `albers_nad83`, or `polar_wgs84`, `dy` gives the grid spacing in meters; if `projection` is `regular_ll`, `dy` gives the grid spacing in degrees. No default value.

8. KNOWN_X : A real value specifying the i-coordinate of an (i,j) location corresponding to a (latitude, longitude) location that is known in the projection. Default value is 1.

9. KNOWN_Y : A real value specifying the j-coordinate of an (i,j) location corresponding to a (latitude, longitude) location that is known in the projection. Default value is 1.

10. KNOWN_LAT : A real value specifying the latitude of a (latitude, longitude) location that is known in the projection. No default value.

11. KNOWN_LON : A real value specifying the longitude of a (latitude, longitude) location that is known in the projection. No default value.

12. STDLON : A real value specifying the longitude that is parallel with the y-axis in conic and azimuthal projections. No default value.

13. TRUELAT1 : A real value specifying, the first true latitude for the Lambert conformal conic projection, or the true latitude for the polar stereographic projection. No default value.

14. TRUELAT2 : A real value specifying, the second true latitude for the Lambert conformal conic projection.. No default value.

15. WORDSIZE : An integer giving the number of bytes used to represent the value of each grid point in the data files. No default value.

16. TILE_X : An integer specifying the number of grid points in the x-direction, *excluding any halo points*, for a single tile of source data. No default value.

17. TILE_Y : An integer specifying the number of grid points in the y-direction, *excluding any halo points*, for a single tile of source data. No default value.

18. TILE_Z : An integer specifying the number of grid points in the z-direction for a single tile of source data; this keyword serves as an alternative to the pair of keywords `tile_z_start` and `tile_z_end`, and when this keyword is used, the starting z-index is assumed to be 1. No default value.

19. TILE_Z_START : An integer specifying the starting index in the z-direction of the array in the data files. If this keyword is used, `tile_z_end` must also be specified. No default value.

20. TILE_Z_END : An integer specifying the ending index in the z-direction of the array in the data files. If this keyword is used, `tile_z_start` must also be specified. No default value

21. CATEGORY_MIN : For categorical data (`type=categorical`), an integer specifying the minimum category index that is found in the data set. If this keyword is used, `category_max` must also be specified. No default value.

22. CATEGORY_MAX : For categorical data (`type=categorical`), an integer specifying the maximum category index that is found in the data set. If this keyword is used, `category_min` must also be specified. No default value.

23. TILE_BDR : An integer specifying the halo width, in grid points, for each tile of data. Default value is 0.

24. MISSING_VALUE : A real value that, when encountered in the data set, should be interpreted as missing data. No default value.

25. SCALE_FACTOR : A real value that data should be scaled by (through multiplication) after being read in as integers from tiles of the data set. Default value is 1.

26. ROW_ORDER : A character string, either `bottom_top` or `top_bottom`, specifying whether the rows of the data set arrays were written proceeding from the lowest-index row to the highest (`bottom_top`) or from highest to lowest (`top_bottom`). This keyword may be useful when utilizing some USGS data sets, which are provided in `top_bottom` order. Default value is `bottom_top`.

27. ENDIAN : A character string, either `big` or `little`, specifying whether the values in the static data set arrays are in big-endian or little-endian byte order. Default value is `big`.

Description of METGRID.TBL Options

The METGRID.TBL file is a text file that defines parameters of each of the meteorological fields to be interpolated by metgrid. Parameters for each field are defined in a separate section, with sections being delimited by a line of equality symbols (e.g., '==========='). Within each section, there are specifications, each of which has the form of *keyword=value*. Some keywords are required in a section, while others are optional; some keywords are mutually exclusive with other keywords. Below, the possible keywords and their expected range of values are described.

1. NAME : A character string giving the name of the meteorological field to which the containing section of the table pertains. The name should exactly match that of the field as given in the intermediate files (and, thus, the name given in the Vtable used in generating the intermediate files). This field is required. No default value.

2. OUTPUT : Either `yes` or `no`, indicating whether the field is to be written to the metgrid output files or not. Default value is `yes`.

3. MANDATORY : Either `yes` or `no`, indicating whether the field is required for successful completion of metgrid. Default value is `no`.

4. OUTPUT_NAME : A character string giving the name that the interpolated field should be output as. When a value is specified for `output_name`, the interpolation options from the table section pertaining to the field with the specified name are used. Thus, the effects of specifying `output_name` are two-fold: The interpolated field is assigned the specified name before being written out, and the interpolation methods are taken from the section pertaining to the field whose name matches the value assigned to the `output_name` keyword. No default value.

5. FROM_INPUT : A character string used to compare against the values in the `fg_name` namelist variable; if `from_input` is specified, the containing table section will only be used when the time-varying input source has a filename that contains the value of

from_input as a substring. Thus, from_input may be used to specify different interpolation options for the same field, depending on which source of the field is being processed. No default value.

6. OUTPUT_STAGGER : The model grid staggering to which the field should be interpolated. For ARW, this must be one of U, V, and M; for NMM, this must be one of HH and VV. Default value for ARW is M; default value for NMM is HH.

7. IS_U_FIELD : Either yes or no, indicating whether the field is to be used as the wind U-component field. For ARW, the wind U-component field must be interpolated to the U staggering (output_stagger=U); for NMM, the wind U-component field must be interpolated to the V staggering (output_stagger=VV). Default value is no.

8. IS_V_FIELD : Either yes or no, indicating whether the field is to be used as the wind V-component field. For ARW, the wind V-component field must be interpolated to the V staggering (output_stagger=V); for NMM, the wind V-component field must be interpolated to the V staggering (output_stagger=VV).Default value is no.

9. INTERP_OPTION : A sequence of one or more names of interpolation methods to be used when horizontally interpolating the field. Available interpolation methods are: average_4pt, average_16pt, wt_average_4pt, wt_average_16pt, nearest_neighbor, four_pt, sixteen_pt, search, average_gcell(*r*) ; for the grid cell average method (average_gcell), the optional argument *r* specifies the minimum ratio of source data resolution to simulation grid resolution at which the method will be applied; unless specified, *r* = 0.0, and the option is used for any ratio. When a sequence of two or more methods are given, the methods should be separated by a + sign. Default value is nearest_neighbor.

10. INTERP_MASK : The name of the field to be used as an interpolation mask, along with the value within that field which signals masked points. A specification takes the form *field*(*maskval*), where *field* is the name of the field and *maskval* is a real value. Default value is no mask.

11. INTERP_LAND_MASK : The name of the field to be used as an interpolation mask when interpolating to water points (determined by the static LANDMASK field), along with the value within that field which signals land points. A specification takes the form *field*(*maskval*), where *field* is the name of the field and *maskval* is a real value. Default value is no mask.

12. INTERP_WATER_MASK : The name of the field to be used as an interpolation mask when interpolating to land points (determined by the static LANDMASK field), along with the value within that field which signals water points. A specification takes the form *field*(*maskval*), where *field* is the name of the field and *maskval* is a real value. Default value is no mask.

13. FILL_MISSING : A real number specifying the value to be assigned to model grid points that received no interpolated value, for example, because of missing or incomplete meteorological data. Default value is 1.E20.

14. Z_DIM_NAME : For 3-dimensional meteorological fields, a character string giving the name of the vertical dimension to be used for the field on output. Default value is `num_metgrid_levels`.

15. DERIVED : Either `yes` or `no`, indicating whether the field is to be derived from other interpolated fields, rather than interpolated from an input field. Default value is `no`.

16. FILL_LEV : The `fill_lev` keyword, which may be specified multiple times within a table section, specifies how a level of the field should be filled if that level does not already exist. A generic value for the keyword takes the form *DLEVEL*:*FIELD*(*SLEVEL*), where *DLEVEL* specifies the level in the field to be filled, *FIELD* specifies the source field from which to copy levels, and *SLEVEL* specifies the level within the source field to use. *DLEVEL* may either be an integer or the string `all`. *FIELD* may either be the name of another field, the string `const`, or the string `vertical_index`. If *FIELD* is specified as `const`, then *SLEVEL* is a constant value that will be used to fill with; if *FIELD* is specified as `vertical_index`, then (*SLEVEL*) must not be specified, and the value of the vertical index of the source field is used; if *DLEVEL* is 'all', then all levels from the field specified by the `level_template` keyword are used to fill the corresponding levels in the field, one at a time. No default value.

17. LEVEL_TEMPLATE : A character string giving the name of a field from which a list of vertical levels should be obtained and used as a template. This keyword is used in conjunction with a `fill_lev` specification that uses `all` in the *DLEVEL* part of its specification. No default value.

18. MASKED : Either `land` or `water`, indicating whether the field is invalid over land or water, respectively. When a field is masked, or invalid, the static LANDMASK field will be used to determine which model grid points the field should be interpolated to; invalid points will be assigned the value given by the `FILL_MISSING` keyword. Default value is null (i.e., the field is valid for both land and water points).

19. MISSING_VALUE : A real number giving the value in the input field that is assumed to represent missing data. No default value.

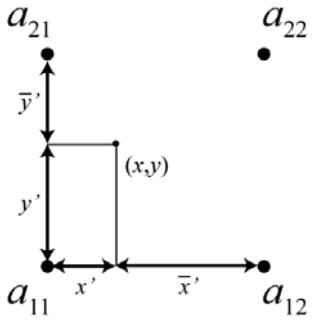
20. VERTICAL_INTERP_OPTION : A character string specifying the vertical interpolation method that should be used when vertically interpolating to missing points. Currently, this option is not implemented. No default value.

21. FLAG_IN_OUTPUT : A character string giving the name of a global attribute which will be assigned a value of 1 and written to the metgrid output if the interpolated field is to be output (`output=yes`). Default value is null (i.e., no flag will be written for the field).

Available Interpolation Options in Geogrid and Metgrid

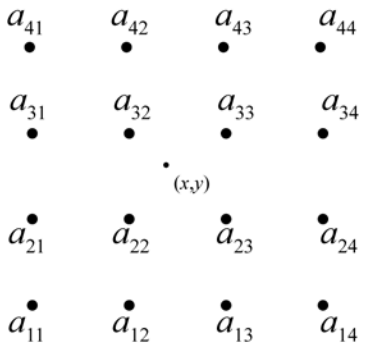
Through the GEOGRID.TBL and METGRID.TBL files, the user can control the method by which source data – either static fields in the case of geogrid or meteorological fields in the case of metgrid – are interpolated. In fact, a list of interpolation methods may be given, in which case, if it is not possible to employ the *i*-th method in the list, the (*i*+1)-st method will be employed, until either some method can be used or there are no methods left to try in the list. For example, to use a four-point bi-linear interpolation scheme for a field, we could specify `interp_option=four_pt`. However, if the field had areas of missing values, which could prevent the `four_pt` option from being used, we could request that a simple four-point average be tried if the `four_pt` method couldn't be used by specifying `interp_option=four_pt+average_4pt` instead. Below, each of the available interpolation options in the WPS are described conceptually; for the details of each method, the user is referred to the source code in the file WPS/geogrid/src/interp_options.F.

1. four_pt : Four-point bi-linear interpolation



The four-point bi-linear interpolation method requires four valid source points a_{ij}, $1 \le i, j \le 2$, surrounding the point (x,y), to which geogrid or metgrid must interpolate, as illustrated in the figure above. Intuitively, the method works by linearly interpolating to the x-coordinate of the point (x,y) between a_{11} and a_{12}, and between a_{21} and a_{22}, and then linearly interpolating to the y-coordinate using these two interpolated values.

2. sixteen_pt : Sixteen-point overlapping parabolic interpolation



The sixteen_pt overlapping parabolic interpolation method requires sixteen valid source points surrounding the point (x,y), as illustrated in the figure above. The method works by fitting one parabola to the points a_{i1}, a_{i2}, and a_{i3}, and another parabola to the points a_{i2}, a_{i3}, and a_{i4}, for row i, $1 \le i \le 4$; then, an intermediate interpolated value p_i within row i at the x-coordinate of the point is computed by taking an average of the values of the two parabolas evaluated at x, with the average being weighted linearly by the distance of x between a_{i2}, and a_{i3}. Finally, the interpolated value at (x,y) is found by performing the same operations as for a row of points, but for the column of interpolated values p_i to the y-coordinate of (x,y).

3. average_4pt : Simple four-point average interpolation

The four-point average interpolation method requires at least one valid source data point from the four source points surrounding the point (x,y). The interpolated value is simply the average value of all valid values among these four points.

4. wt_average_4pt : Weighted four-point average interpolation

The weighted four-point average interpolation method can handle missing or masked source data points, and the interpolated value is given as the weighted average of all valid values, with the weight w_{ij} for the source point a_{ij}, $1 \le i, j \le 2$, given by

$$w_{ij} = \max\{0, 1 - \sqrt{(x - x_i)^2 + (y - y_j)^2}\}.$$

Here, x_i is the x-coordinate of a_{ij} and y_j is the y-coordinate of a_{ij}.

5. average_16pt : Simple sixteen-point average interpolation

The sixteen-point average interpolation method works in an identical way to the four-point average, but considers the sixteen points surrounding the point (x,y).

6. wt_average_16pt : Weighted sixteen-point average interpolation

The weighted sixteen-point average interpolation method works like the weighted four-point average, but considers the sixteen points surrounding (*x*,*y*); the weights in this method are given by

$$w_{ij} = \max\{0, 2 - \sqrt{(x - x_i)^2 + (y - y_j)^2}\},$$

where x_i and y_j are as defined for the weighted four-point method, and $1 \le i, j \le 4$.

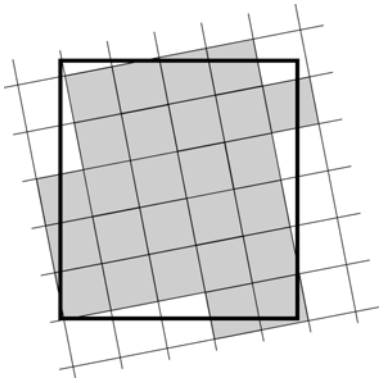
7. nearest_neighbor : Nearest neighbor interpolation

The nearest neighbor interpolation method simply sets the interpolated value at (*x*,*y*) to the value of the nearest source data point, regardless of whether this nearest source point is valid, missing, or masked.

8. search : Breadth-first search interpolation

The breadth-first search option works by treating the source data array as a 2-d grid graph, where each source data point, whether valid or not, is represented by a vertex. Then, the value assigned to the point (*x*,*y*) is found by beginning a breadth-first search at the vertex corresponding to the nearest neighbor of (*x*,*y*), and stopping once a vertex representing a valid (i.e., not masked or missing) source data point is found.

9. average_gcell : Model grid-cell average



The grid-cell average interpolator may be used when the resolution of the source data is higher than the resolution of the model grid. For a model grid cell Γ, the method takes a simple average of the values of all source data points that are nearer to the center of Γ

than to the center of any other grid cell. The operation of the grid-cell average method is illustrated in the figure above, where the interpolated value for the model grid cell – represented as the large rectangle – is given by the simple average of the values of all of the shaded source grid cells.

Land Use and Soil Categories in the Static Data

The default land use and soil category data sets that are provided as part of the WPS static data tar file contain categories that are matched with the USGS categories described in the VEGPARM.TBL and SOILPARM.TBL files in the WRF run directory. Descriptions of the 24 land use categories and 16 soil categories are provided in the tables below.

Table 1: USGS 24-category Land Use Categories

Land Use Category	Land Use Description
1	Urban and Built-up Land
2	Dryland Cropland and Pasture
3	Irrigated Cropland and Pasture
4	Mixed Dryland/Irrigated Cropland and Pasture
5	Cropland/Grassland Mosaic
6	Cropland/Woodland Mosaic
7	Grassland
8	Shrubland
9	Mixed Shrubland/Grassland
10	Savanna
11	Deciduous Broadleaf Forest
12	Deciduous Needleleaf Forest
13	Evergreen Broadleaf
14	Evergreen Needleleaf
15	Mixed Forest
16	Water Bodies
17	Herbaceous Wetland
18	Wooden Wetland
19	Barren or Sparsely Vegetated
20	Herbaceous Tundra
21	Wooded Tundra
22	Mixed Tundra
23	Bare Ground Tundra
24	Snow or Ice

Table 2: 16-category Soil Categories

Soil Category	Soil Description
1	Sand
2	Loamy Sand
3	Sandy Loam
4	Silt Loam
5	Silt
6	Loam
7	Sandy Clay Loam
8	Silty Clay Loam
9	Clay Loam
10	Sandy Clay
11	Silty Clay
12	Clay
13	Organic Material
14	Water
15	Bedrock
16	Other (land-ice)

Chapter 4: WRF Initialization

Table of Contents

Introduction

The WRF model has two large classes of simulations that it is able to generate: those with an *ideal* initialization and those utilizing *real* data. The idealized simulations typically manufacture an initial condition file for the WRF model from an existing 1-D or 2-D sounding and assume a simplified orography. The real-data cases usually require pre-processing from the WPS package, which provides atmospheric and static fields with a fidelity appropriate to the chosen grid resolution for the model. The WRF model itself is not altered by choosing one initialization option over another, but the WRF pre-processors, the real.exe and ideal.exe programs, are specifically built based upon a user's selection.

The real.exe and ideal.exe programs are never used together. Both the real.exe and ideal.exe are the programs that are processed just prior to the WRF model run.

The ideal *vs* real cases are divided as follows:

- Ideal cases – initialization programs named “ideal.exe”
  - 3d
    - em_b_wave - baroclinic wave, 100 km
    - em_heldsuarez – global case with polar filtering, 625 km
    - em_les – large eddy simulation, 100 m
    - em_quarter_ss - super cell, 2 km
  - 2d
    - em_grav2d_x – gravity current, 100 m
    - em_hill2d_x – flow over a hill, 2 km
    - em_seabreeze – water and land, 2 km, full physics
    - em_squall2d_x – squall line, 250 m
    - em_squall2d_y – transpose of above problem

- Real data cases – initialization program named “real.exe”
  - em_real – examples from 4 to 30 km

The selection of the type of forecast is made when issuing the **`./compile`** statement. When selecting a different case to study, the code must be re-compiled to choose the correct initialization model. For example, after configuring the setup for the architecture, if the user issues the command **`./compile em_real`**, then the initialization program is built using **`module_initialize_real.F`** as the target module (one of the **`./WRFV3/dyn_em/module_initialize_*.F`** files). Similarly, if the users specifies **`./compile em_les`**, then the Fortran module for the large eddy simulation (**`module_initialize_les.F)`** is automatically inserted into the build for ideal.exe. In each of these initialization modules, the same sort of activities goes on:

- compute a base state / reference profile for geopotential and column pressure
- compute the perturbations from the base state for geopotential and column pressure
- initialize meteorological variables: u, v, potential temperature, vapor mixing ratio
- initialize static fields for the projection and the physical surface; for many of the idealized cases, these are simplified initializations such as map factors set to one, and topography elevation set to zero

Both the real.exe program and ideal.exe programs share a large portion of source code, to handle the following duties:

- read data from the namelist
- allocate space
- generate initial condition file

The real-data case does some additional processing:

- read input data from the WRF Preprocessing System (WPS)
- compute dry surface pressure, model levels, and vertically interpolate data
- prepare soil fields for use in model (usually, vertical interpolation to the requested levels)
- check to verify soil categories, land use, land mask, soil temperature, sea surface temperature are all consistent with each other
- multiple input time periods are processed to generate the lateral boundary conditions, which are required unless processing a global forecast
- 3d boundary data (u, v, potential temperature, vapor mixing ratio, total geopotential) are coupled with map factors (on the correct staggering) and total column pressure

The “real.exe” program may be run as either a serial or a distributed memory job. Since the idealized cases only require that the initialization run for a single time period (no lateral boundary file is required) and are therefore quick to process, all of the “ideal.exe” programs should be run on a single processor as a serial job.

Initialization for Ideal Cases

The program "ideal.exe" is the program in the WRF system to run with a controlled scenario. Typically this program requires no input except for the namelist.input and the input_sounding files (except for the b_wave case which uses a 2-D binary sounding file). The program outputs the wrfinput_d01 file that is read by the WRF model executable ("wrf.exe") . Since no external data is required to run the idealized cases, even for researchers interested in real-data cases, the idealized simulations are an easy way to insure that the model is working correctly on a particular architecture and compiler.

Idealized runs can use any of the boundary conditions except "specified", and are not, by default, set up to run with sophisticated physics (other than from microphysics). Most have are no radiation, surface fluxes or frictional effects (other than the sea breeze case, LES, and the global Held-Suarez). The idealized cases are mostly useful for dynamical studies, reproducing converged or otherwise known solutions, and idealized cloud modeling.

There are 2d and 3d examples of idealized cases, with and without topography, and with and without an initial thermal perturbation. The namelist can control the size of domain, number of vertical levels, model top height, grid size, time step, diffusion and damping properties, boundary conditions, and physics options. A large number of existing namelist settings are already found within each of the directories associated with a particular case.

The input_sounding (also already in appropriate case directories) can be any set of levels that goes at least up to the model top height (ztop) in the namelist. The first line is the surface pressure (hPa), potential temperature (K) and moisture mixing ratio (g/kg). Each subsequent line has five input values: height (meters above sea-level), potential temperature (K), vapor mixing ratio (g/kg), x-direction wind component (m/s), y-direction wind component (m/s). The “ideal.exe” program interpolates the data from the input_sounding file, and will extrapolate if not enough data is provided.

The base state sounding for idealized cases is the initial sounding minus the moisture, and so does not have to be defined separately. Note for the baroclinic wave case: a 1-D input sounding is not used because the initial 3d arrays are read in from the file input_jet. This means for the baroclinic wave case the namelist.input file cannot be used to change the horizontal or vertical dimensions.

Making modifications apart from namelist-controlled options or soundings has to be done by editing the Fortran code. Such modifications would include changing the topography, the distribution of vertical levels, the properties of an initialization bubble, or preparing a case to use more physics, such as a land-surface model. The Fortran code to edit is contained in ./WRFV3/dyn_em/module_initialize_[case].F, where [case] is the case chosen in compilation, e.g. module_initialize_squall2d_x.F. The subroutine to modify is init_domain_rk. To change the vertical levels, only the 1d array znw must be defined, containing the full levels starting from 1 at k=1 and ending with 0 at k=kde. To change

the topography, only the 2d array ht(i,j) must be defined, making sure it is periodic if those boundary conditions are used. To change the bubble, search for the string "bubble" to locate the code to change.

Each of the cases provide an excellent set of examples to the user. The method to specify a thermal bubble is given in the super cell case. In the hill2d case, the topography is accounted for properly in setting up the initial 3d arrays, so that example should be followed for any topography cases. A symmetry example in the squall line cases tests that your indexing modifications are correct.

Available Ideal Test Cases

The available test cases are

1. squall2d_x (test/em_squall2d_x)
    - 2D squall line (x,z) using Kessler microphysics and a fixed 300 m^2/s viscosity.
    - periodicity condition used in y so that 3D model produces 2D simulation.
    - v velocity should be zero and there should be no variation in y in the results.
2. squall2d_y (test/em_squall2d_y)
    - Same as squall2d_x, except with (x) rotated to (y).
    - u velocity should be zero and there should be no variation in x in the results.
3. 3D quarter-circle shear supercell simulation (test/em_quarter_ss).
    - Left and right moving supercells are produced.
    - See the README.quarter_ss file in the test directory for more information.
4. 2D flow over a bell-shaped hill (x,z) (test/em_hill2d_x)
    - 10 km half-width, 2 km grid-length, 100 m high hill, 10 m/s flow, N=0.01/s, 30 km high domain, 80 levels, open radiative boundaries, absorbing upper boundary.
    - Case is in linear hydrostatic regime, so vertical tilted waves with ~6km vertical wavelength.
5. 3D baroclinic waves (test/em_b_wave)
    - Baroclinically unstable jet u(y,z) on an f-plane.
    - Symmetric north and south, periodic east and west boundaries.
    - 100 km grid size 16 km top with 4 km damping layer.
    - 41x81 points in (x,y), 64 layers.
6. 2D gravity current (test/em_grav2d_x)
    - Test case is described in Straka et al, *INT J NUMER METH FL* **17** (1): 1-22 JUL 15 1993.
    - See the README.grav2d_x file in the test directory.
7. 2D sea breeze (test/em_seabreeze_x)
    - 2 km grid size, 20 km top, land/ water.

    - Can be run with full physics, radiation, surface, boundary layer, land options.
8. 3D large eddy simulation (test/em_les)
    - 100 m grid size, 2 km top.
    - Surface layer physics with fluxes.
    - Doubly periodic
9. 3D Held-Suarez (test/em_heldsuarez)
    - global domain, 625 km in x-direction, 556 km in y-direction, 120 km top.
    - Radiation, polar filter above 45^{o}.
    - Period in x-direction, polar boundary conditions in y-direction

Initialization for Real Data Cases

The real-data WRF cases are those that have the input data to the “real.exe” program provided by the WRF Preprocessing System (WPS). This data from the WPS was originally generated from a previously run external analysis or forecast model. The original data was probably in GriB format and was probably ingested into the WPS by first ftp'ing the raw GriB data from one of the national weather agencies’ anonymous ftp sites.

For example, suppose a WRF forecast is desired with the following criteria:

- 2000 January 24 1200 UTC through January 25 1200 UTC
- the original GriB data is available at 6 h increments

The following files will be generated by the WPS:

- met_em.d01.2000-01-24_12:00:00
- met_em.d01.2000-01-24_18:00:00
- met_em.d01.2000-01-25_00:00:00
- met_em.d01.2000-01-25_06:00:00
- met_em.d01.2000-01-25_12:00:00

The convention is to use "met" to signify data that is output from the WPS “metgrid.exe” program and input into the “real.exe” program. The "d01" portion of the name identifies to which domain this data refers, which permits nesting. The trailing characters are the date, where each WPS output file has only a single time-slice of processed data. For regional forecasts, multiple time periods must be processed by “real.exe” so that a lateral boundary file is available to the model. The global option for WRF requires only an initial condition.

The WPS package delivers data that is ready to be used in the WRF system by the “real.exe” program.

- The data adheres to the WRF IO API. Unless you are developing special tools, stick with the netCDF option to communicate between the WPS package and "real.exe".
- The data has already been horizontally interpolated to the correct grid-point staggering for each variable, and the winds are correctly rotated to the WRF model projection.
- 3D meteorological data from the WPS: pressure, u, v, potential temperature, vapor mixing ratio, geopotential height
- 3D surface data from the WPS: soil temperature, soil moisture, soil liquid
- 2D meteorological data from the WPS: land sea mask, sea level pressure, sea surface temperature, sea ice, snow depth, water equivalent snow depth, canopy water
- 2D static data for the physical surface: terrain elevation, land use categories, soil texture categories, temporally interpolated monthly data, elevation of the input model's topography
- 2D static data for the projection: map factors, Coriolis, projection rotation
- 1D array of the vertical coordinate
- constants: domain size, date

Real Data Test Case: 2000 January 24/12 through 25/12

- A test data set is accessible from the WRF download page. Under the "WRF Model Test Data" list, select the January data. This is a 74x61, 30-km domain centered over the eastern US.
- make sure you have successfully built the code (fine-grid nested initial data is available in the download, so the code may be built with the basic nest option), ./WRFV3/main/real.exe and ./WRFV3/main/wrf.exe both exist
- in the ./WRFV3/test/em_real directory, copy the namelist for the January case to the default name (**`cp namelist.input.jan00 namelist.input`**)
- link the WPS files (the "met_em*" files from the download) into the ./WRFV3/test/em_real directory
- for a single processor, to execute the real program, type **`real.exe`** (this should take less than a minute for this small case with five time periods)
- after running the real.exe program, the files "wrfinput_d01" and "wrfbdy_d01" should be in the directory, these will be directly used by the WRF model
- the "wrf.exe" program is executed next (type **`wrf.exe`**), this should take a few minutes (only a 12 h forecast is requested in the namelist file)
- the output file wrfout_d01:2000-01-24_12:00:00 should contain a 12-h forecast at 3-h intervals

Chapter 5: WRF Model

Table of Contents

Introduction

The WRF model is a fully compressible, and nonhydrostatic model (with a runtime hydrostatic option). Its vertical coordinate is a terrain-following hydrostatic pressure coordinate. The grid staggering is the Arakawa C-grid. The model uses the Runge-Kutta 2nd and 3rd order time integration schemes, and 2nd to 6th order advection schemes in both horizontal and vertical. It uses a time-split small step for acoustic and gravity-wave modes. The dynamics conserves scalar variables.

The WRF model code contains several initialization programs (*ideal.exe* and *real.exe*; see Chapter 4), a numerical integration program (*wrf.exe*), and a program to do one-way nesting (*ndown.exe*). The WRF model Version 3 supports a variety of capabilities. These include

- Real-data and idealized simulations
- Various lateral boundary condition options for real-data and idealized simulations
- Full physics options, and various filter options

- Positive-definite advection scheme
- Non-hydrostatic and hydrostatic (runtime option)
- One-way, two-way nesting and moving nest
- Three-dimensional analysis nudging
- Observation nudging
- Regional and global applications
- Digital filter initialization

Other References

- WRF tutorial presentation: http://www.mmm.ucar.edu/wrf/users/supports/tutorial.html
- WRF-ARW Tech Note (V3 will be available in summer 2008): http://www.mmm.ucar.edu/wrf/users/pub-doc.html
- Chapter 2 of this document for software requirement.

Installing WRF

Before compiling WRF code on a computer, check to see if the netCDF library is installed. This is because one of the supported WRF I/O options is netCDF, and it is the one commonly used, and supported by the post-processing programs. If the netCDF is installed in a path other than `/usr/local/`, then find the path, and use the environment variable NETCDF to define where the path is. To do so, type

```
setenv NETCDF path-to-netcdf-library
```

Often the netCDF library and its include/ directory are collocated. If this is not the case, create a directory, link both netCDF lib and include directories in this directory, and use environment variable to set the path to this directory.

If the netCDF library is not available on the computer, it needs to be installed first. NetCDF source code or pre-built binary may be downloaded from and installation instruction can be found on the Unidata Web page at http://www.unidata.ucar.edu/.

***Hint*: for Linux users:**

If PGI or Intel compiler are used on a Linux computer, make sure netCDF is installed using the same compiler. Use NETCDF environment variable to point to the PGI/Intel compiled netCDF library.

WRF source code tar file can be downloaded from http://www.mmm.ucar.edu/wrf/download/get_source.html. Once the tar file is gunzipped (`gunzip WRFV3.TAR.gz`), and untared (`untar WRFV3.TAR`), and it will create a WRFV3/ directory. This contains:

`Makefile`	Top-level makefile
`README`	General information about WRF/ARW core
`README_test_cases`	Explanation of the test cases
`README.NMM`	General information for WRF/NMM core
`README.rsl_output`	For NMM
`Registry/`	Directory for WRF Registry files
`arch/`	Directory where compile options are gathered
`clean`	script to clean created files, executables
`compile`	script for compiling WRF code
`configure`	script to create the *configure.wrf* file for compile
`chem/`	WRF chemistry, supported by NOAA/GSD
`dyn_em/`	Directory for ARW dynamics and numerics
`dyn_exp/`	Directory for a 'toy' dynamic core
`dyn_nmm/`	Directory for NMM dynamics and numerics, supported by DTC
`external/`	Directory that contains external packages, such as those for IO, time keeping and MPI
`frame/`	Directory that contains modules for WRF framework
`inc/`	Directory that contains include files
`main/`	Directory for main routines, such as wrf.F, and all executables after compilation
`phys/`	Directory for all physics modules
`run/`	Directory where one may run WRF
`share/`	Directory that contains mostly modules for WRF mediation layer and WRF I/O
`test/`	Directory that contains test case directories, may be used to run WRF
`tools/`	Directory that contains tools for developers

The steps to compile and run the model are:

1. configure: generate a configuration file for compilation
2. compile: compile the code
3. run the model

Go to WRFV3 (top) directory and type

```
./configure
```

and a list of choices for your computer should appear. These choices range from compiling for a single processor job (serial), to using OpenMP shared-memory (smpar) or distributed-memory parallelization (dmpar) options for multiple processors, or combination of shared-memory and distributed memory options (dm+sm). When a selection is made, a second choice for compiling nesting will appear. For example, on a Linux computer, the above steps look like:

```
> setenv NETCDF /usr/local/netcdf-pgi
> ./configure

checking for perl5... no
checking for perl... found /usr/bin/perl (perl)
Will use NETCDF in dir: /usr/local/netcdf-pgi
PHDF5 not set in environment. Will configure WRF for use without.
$JASPERLIB or $JASPERINC not found in environment, configuring to build
without grib2 I/O...
------------------------------------------------------------------------
Please select from among the following supported platforms.

1.  Linux i486 i586 i686, gfortran compiler with gcc  (serial)
2.  Linux i486 i586 i686, gfortran compiler with gcc  (smpar)
3.  Linux i486 i586 i686, gfortran compiler with gcc  (dmpar)
4.  Linux i486 i586 i686, gfortran compiler with gcc  (dm+sm)
5.  Linux i486 i586 i686, g95 compiler with gcc  (serial)
6.  Linux i486 i586 i686, g95 compiler with gcc  (dmpar)
7.  Linux i486 i586 i686, PGI compiler with gcc  (serial)
8.  Linux i486 i586 i686, PGI compiler with gcc  (smpar)
9.  Linux i486 i586 i686, PGI compiler with gcc  (dmpar)
10.  Linux i486 i586 i686, PGI compiler with gcc  (dm+sm)
11.  Linux x86_64 i486 i586 i686, ifort compiler with icc (non-SGI
installations)  (serial)
12.  Linux x86_64 i486 i586 i686, ifort compiler with icc (non-SGI
installations)  (smpar)
13.  Linux x86_64 i486 i586 i686, ifort compiler with icc (non-SGI
installations)  (dmpar)
14.  Linux x86_64 i486 i586 i686, ifort compiler with icc (non-SGI
installations)  (dm+sm)
15.  Linux i486 i586 i686 x86_64, PathScale compiler with pathcc
(serial)
16.  Linux i486 i586 i686 x86_64, PathScale compiler with pathcc
(dmpar)

Enter selection [1-16] : 9

Compile for nesting? (0=no nesting, 1=basic, 2=preset moves, 3=vortex
following) [default 0]: 1
```

Enter appropriate options that are best for your computer and application.

When the return key is hit, a `configure.wrf` file will be created. Edit compile options/paths, if necessary.

Hint: It is helpful to start with something simple, such as the serial build. If it is successful, move on to build smpar or dmpar code. Remember to type '`clean -a`' between each build.

Hint: On some computers (e.g. some Intel machines), it may be necessary to set the following environment variable before one compiles:

```
setenv WRF_EM_CORE 1
```

To compile the code, type

```
./compile
```

and the following choices will appear:

```
Usage:

compile wrf           compile wrf in run dir (Note, no
real.exe, ndown.exe or ideal.exe generated)

or choose a test case (see README_test_cases for
details):

   compile em_b_wave
   compile em_esmf_exp (example only)
   compile em_grav2d_x
   compile em_heldsuarez
   compile em_hill2d_x
   compile em_les
   compile em_quarter_ss
   compile em_real
   compile em_seabreeze2d_x
   compile em_squall2d_x
   compile em_squall2d_y
   compile exp_real (example of a toy solver)
   compile nmm_real (NMM solver)

 compile -h              help message
```

where **em** stands for the Advanced Research WRF dynamic solver (which currently is the '**E**ulerian **m**ass-coordinate' solver). Type one of the above to compile. When you switch from one test case to another, you must type one of the above to recompile. The recompile is necessary to create a new initialization executable (i.e. `real.exe`, and `ideal.exe` - there is a different `ideal.exe` for each of the idealized test cases), while `wrf.exe` is the same for all test cases.

If you want to remove all object files (except those in `external/`directory) and executables, type '*clean*'.

Type '*clean -a*' to remove built files in ALL directories, including `configure.wrf`. This is recommended if you make any mistake during the process, or if you have edited the `Registry.EM` file.

a. Idealized case

For any 2D test cases (labeled in the case names), serial or OpenMP (smpar) compile options must be used. Suppose you would like to compile and run the 2-dimensional squall case, type

```
./compile em_squall2d_x >& compile.log
```

After a successful compilation, you should have two executables created in the **`main`**/ directory: **`ideal.exe`** and **`wrf.exe`**. These two executables will be linked to the corresponding `test/case_name` and `run`/ directories. `cd` to either directory to run the model.

It is a good practice to save the entire compile output to a file. When the executables were not present, this output is useful to help diagnose the compiler errors.

b. Real-data case

For a real-data case, type

```
./compile em_real >& compile.log &
```

When the compile is successful, it will create three executables in the **`main`**/directory: `ndown.exe`, `real.exe` and `wrf.exe`.

`real.exe`: for WRF initialization of real data cases
`ndown.exe` : for one-way nesting
`wrf.exe` : WRF model integration

Like in the idealized cases, these executables will be linked to `test/em_real` and `run`/ directories. `cd` to one of these two directories to run the model.

Running WRF

One may run the model executables in either the `run`/ directory, or the `test/case_name` directory. In either case, one should see executables, `ideal.exe` or `real.exe` (and `ndown.exe`), and `wrf.exe`, linked files (mostly for real-data cases), and one or more `namelist.input` files in the directory.

Hint: If you would like to run the model executables in a different directory, copy or link the files in `test/em_*` directory to that directory, and run from there.

Idealized, real data, restart run, two-way nested, and one-way nested runs are explained on the following pages. Read on.

a. Idealized case

Suppose the test case `em_squall2d_x` is compiled, to run, type

`cd test/em_``squall2d_x`

Edit `namelist.input` file (see `README.namelist` in `WRFV3/run/` directory or its Web version) to change length of integration, frequency of output, size of domain, timestep, physics options, and other parameters.

If you see a script in the test case directory, called `run_me_first.csh`, run this one first by typing:

```
./run_me_first.csh
```

This links some physics data files that might be needed to run the case.

To run the initialization program, type

```
./ideal.exe
```

This program will typically read an input sounding file located in that directory, and generate an initial condition file `wrfinput_d01`. All idealized cases do not require lateral boundary file because of the boundary condition choices they use, such as the periodic option. If the job is run successfully, the last thing it prints should be: '`wrf: SUCCESS COMPLETE IDEAL INIT`'.

To run the model and save the standard output to a file, type

```
./wrf.exe >& wrf.out &
```

or for a 3D test case compiled with MPI (dmpar) option,

```
mpirun -np 4 ./wrf.exe
```

Pairs of `rsl.out.*` and `rsl.error.*` files will appear with any MPI runs. These are standard out and error files. Note that the execution command for MPI runs may be different on different machines. Check the user manual.

If the model run is successful, the last thing printed in 'wrf.out' or rsl.*.0000 file should be: 'wrf: SUCCESS COMPLETE WRF'. Ouput files wrfout_d01_0001-01-01* and wrfrst* should be present in the run directory, depending on how namelist variables are specified for output. The time stamp on these files originates from the start times in the namelist file.

b. Real-data case

To make a real-data case run, cd to the working directory by typing

```
cd test/em_real (or cd run)
```

Start with a namelist.input template file in the directory, edit it to match your case.

Running a real-data case requires successfully running the **WRF Preprocessing System** programs (or WPS). Make sure met_em.* files from WPS are seen in the run directory (either link or copy the files):

```
ls -l ../../WPS/met_em*
ln -s ../../WPS/met_em* .
```

Make sure you edit the following variables in namelist.input file:

num_metgrid_levels: number of_ incoming data levels (can be found by using ncdump command on met_em.d01.<date> file)
eta_levels: model *eta* levels from 1 to 0, if you choose to do so. If not, real will compute a nice set of *eta* levels.

Other options for use to assist vertical interpolation are:

use_surface: whether to use surface input data
extrap_type: vertical extrapolation of non-temperature fields
t_extrap_type: vertical extrapolation for potential temperature
use_levels_below_ground: use levels below input surface level
force_sfc_in_vinterp: force vertical interpolation to use surface data
lowest_lev_from_sfc: place surface data in the lowest model level
p_top_requested: pressure top used in the model, default is 5000 Pa
interp_type: vertical interpolation method: linear in p(default) or log(p)
lagrange_order: vertical interpolation order, linear (default) or quadratic
zap_close_levels: allow surface data to be used if it is close to a constant pressure level.

Other minimum set of namelist variables to edit are:

`start_*, end_*:` start and end times for data processing and model integration
`interval_seconds:` input data interval for boundary conditions
`time_step:` model time step, and can be set as large as 6*DX (in km)
`e_ws, e_sn, e_vert:` domain dimensions in west-east, south-north and vertical
`dx, dy:` model grid distance in meters

To run real-data initialization program compiled using serial or OpenMP (smpar) options, type

```
./real.exe >& real.out
```

Successful completion of the job should have '`real_em: SUCCESS EM_REAL INIT`' printed at the end of `real.out` file. It should also produce `wrfinput_d01` and `wrfbdy_d01` files. In real data case, both files are required.

Run WRF model by typing

```
./wrf.exe
```

A successful run should produce one or several output files named like `wrfout_d01_yyyy-mm-dd_hh:mm:ss`. For example, if you start the model at 1200 UTC, January 24 2000, then your first output file should have the name:

```
wrfout_d01_2000-01-24_12:00:00
```

The time stamp on the file name is always the first time the output file is written. It is always good to check the times written to the output file by typing:

```
ncdump -v Times wrfout_d01_2000-01-24_12:00:00
```

You may have other wrfout files depending on the namelist options (how often you split the output files and so on using namelist option `frames_per_outfile`).You may also create restart files if you have restart frequency (`restart_interval` in the namelist.input file) set within your total integration length. The restart file should have names like

```
wrfrst_d01_yyyy-mm-dd_hh:mm:ss
```

The time stamp on a restart file is the time that restart file is valid at.

For DM (distributed memory) parallel systems, some form of **mpirun** command will be needed to run the executables. For example, on a Linux cluster, the command to run MPI code and using 4 processors may look like:

```
mpirun -np 4 ./real.exe
mpirun -np 4 ./wrf.exe
```

On some IBMs, the command may be:

```
poe ./real.exe
poe ./wrf.exe
```

for a batch job, and

```
poe ./real.exe -rmpool 1 -procs 4
poe ./wrf.exe -rmpool 1 -procs 4
```

for an interactive run. (Interactive MPI job is not an option on NCAR IBMs *bluevista* and *blueice*)

c. Restart Run

A restart run allows a user to extend a run to a longer simulation period. It is effectively a continuous run made of several shorter runs. Hence the results at the end of one or more restart runs should be identical to a single run without any restart.

In order to do a restart run, one must first create restart file. This is done by setting namelist variable `restart_interval` (unit is in minutes) to be equal to or less than the simulation length in the first model run, as specified by `run_*` variables or `start_*` and `end_*` times. When the model reaches the time to write a restart file, a restart file named `wrfrst_<domain_id>_<date>` will be written. The date string represents the time when the restart file is valid.

When one starts the restart run, edit the `namelist.input` file, so that your `start_*` time will be set to the restart time (which is the time the restart file is written). The other namelist variable one must set is `restart`, this variable should be set to .true. for a restart run.

In summary, these namelists should be modified:

`start_*, end_*:`	start and end times for restart model integration
`restart:`	logical to indicate whether the run is a restart or not

d. Two-way Nested Runs

A two-way nested run is a run where multiple domains at different grid resolutions are run simultaneously and communicate with each other: The coarser domain provides boundary values for the nest, and the nest feedbacks its calculation back to the coarser domain. The model can handle multiple domains at the same nest level (no overlapping nest), and multiple nest levels (telescoping).

When preparing for a nested run, make sure that the code is compiled with basic nest options (option 1).

Most of options to start a nest run are handled through the namelist. ***All variables in the*** `namelist.input` ***file that have multiple columns of entries need to be edited with caution***. Do start with a namelist template. The following are the key namelist variables to modify:

`start_*, end_*`: start and end simulation times for the nest

`input_from_file`: whether a nest requires an input file (e.g. `wrfinput_d02`). This is typically used for a real data case, since the nest input file contains nest topography and land information.

`fine_input_stream`: which fields from the nest input file are used in nest initialization. The fields to be used are defined in the Registry.EM. Typically they include static fields (such as terrain, landuse), and masked surface fields (such as skin temperature, soil moisture and temperature). Useful for nest starting at a later time than the coarse domain.

`max_dom`: the total number of domains to run. For example, if you want to have one coarse domain and one nest, set this variable to 2.

`grid_id`: domain identifier that is used in the wrfout naming convention. The most coarse grid must have grid_id of 1.

`parent_id`: used to indicate the parent domain of a nest. `grid_id` value is used.

`i_parent_start/j_parent_start`: lower-left corner starting indices of the nest domain in its parent domain. These parameters should be the same as in `namelist.wps`.

`parent_grid_ratio`: integer parent-to-nest domain grid size ratio. Typically odd number ratio is used in real-data applications.

`parent_time_step_ratio`: integer time-step ratio for the nest domain. It may be different from the `parent_grid_ratio`, though they are typically set the same.

`feedback`: this is the key setup to define a two-way nested (or one-way nested) run. When feedback is on, the values of the coarse domain are overwritten by the values of the variables (average of cell values for mass points, and average of the cell-face values for horizontal momentum points) in the nest at the coincident points. For masked fields, only the single point value at the collocating points is fedback. If the `parent_grid_ratio` is even, an arbitrary choice of southwest corner point value is used for feedback. This is the reason it is better to use odd `parent_grid_ratio` with this option. When feedback is off , it is equivalent to a one-way nested run, since nest results are not reflected in the parent domain.

`smooth_option`: this a smoothing option for the parent domain in area of the nest if *feedback* is on. Three options are available: 0 = no smoothing; 1 = 1-2-1 smoothing; 2 = smoothing-desmoothing.

3-D Idealized Cases

For 3-D idealized cases, no nest input files are required. The key here is the specification of the `namelist.input` file. What the model does is to interpolate all variables required in the nest from the coarse domain fields. Set

```
input_from_file = F, F
```

Real Data Cases

For real-data cases, three input options are supported. The first one is similar to running the idealized cases. That is to have all fields for the nest interpolated from the coarse domain (`input_from_file = T, F`). The disadvantage of this option is obvious, one will not benefit from the higher resolution static fields (such as terrain, landuse, and so on).

The second option is to set `input_from_file = T` for each domain, which means that the nest will have a nest wrfinput file to read in. The limitation of this option is that this only allows the nest to start at the same time as the coarse domain.

The third option is in addition to setting `input_from_file = T` for each domain, also set `fine_input_stream = 2` for each domain. Why a value of 2? This is based on the Registry setting, which designates certain fields to be read in from auxiliary input stream number 2. This option allows the nest initialization to use 3-D meteorological fields interpolated from the coarse domain, static fields and masked, time-varying surface fields from the nest wrfinput. It hence allows a nest to start at a later time than hour 0. Setting `fine_input_stream = 0` is equivalent to the second option.

To run `real.exe` for a nested run, one must first run WPS and create data for all the nests. Suppose WPS is run for a two-domain nest case, and these files should be present in a WPS directory:

```
met_em.d01.2000-01-24_12:00:00
met_em.d01.2000-01-24_18:00:00
met_em.d01.2000-01-25_00:00:00
met_em.d01.2000-01-25_06:00:00
met_em.d01.2000-01-25_12:00:00
met_em.d02.2000-01-24_12:00:00
```

Typically only the first time period of the nest input file is needed to create nest wrfinput file. Link or move all these files to the run directory.

Edit the `namelist.input` file and set the correct values for all relevant variables, described on the previous pages (in particular, set `max_dom = 2`, for the total number of domains to run), as well as physics options. Type the following to run:

```
./real.exe >& real.out
```

or

```
mpirun -np 4 ./real.exe
```

If successful, this will create all input files for coarse as well as nest domains. For a two-domain example, these are

```
wrfinput_d01
wrfinput_d02
wrfbdy_d01
```

To run WRF, type

```
./wrf.exe
```

or

```
mpirun -np 4 ./wrf.exe
```

If successful, the model should create wrfout files for both domain 1 and 2:

```
wrfout_d01_2000-01-24_12:00:00
wrfout_d02_2000-01-24_12:00:00
```

e. One-way Nested Run Using *ndown*

WRF supports two separate one-way nested option. In this section, one-way nesting is defined as a finer-grid-resolution run made as a subsequent run after the coarser-grid-resolution run, where the `ndown` program is run in between the two forecasts. The initial and lateral boundary conditions for this finer-grid run are obtained from the coarse grid run, together with input from higher resolution terrestrial fields (e.g. terrain, landuse, etc.), and masked surface fields (such as soil temperature and moisture). The program that performs this task is `ndown.exe`. Note that the use of this program requires the code to be compiled for nesting.

When one-way nesting is used, the coarse-to-fine grid ratio is only restricted to be an integer. An integer less than or equal to 5 is recommended.

To make a one-way nested run involves these steps:

1) Generate a coarse-grid model output
2) Make temporary fine-grid initial condition `wrfinput_d01` file (note that only a single time period is required, valid at the desired start time of the fine-grid domain)
3) Run program `ndown`**,** with coarse-grid model output and a fine-grid initial

condition to generate fine grid initial and boundary conditions, similar to the output from the `real.exe` program)
4) Run the fine-grid simulation

To compile, choose an option that supports nesting.

Step 1: Make a coarse grid run

This is no different than any of the single domain WRF run as described above.

Step 2: Make a temporary fine grid initial condition file

The purpose of this step is to ingest higher resolution terrestrial fields and corresponding land-water masked soil fields.

Before doing this step, WPS should be run for one coarse and one nest domains (this helps to line up the nest with the coarse domain), and for the one time period the one-way nested run is to start. This generates a WPS output file for the nested domain (domain 2): `met_em.d02.<date>`.

- Rename `met_em.d02.*` to `met.d01.*` for the single requested fine-grid start time. Move the original domain 1 WPS output files before you do this.
- Edit the `namelist.input` file for fine-grid domain (pay attention to column 1 only) and edit in the correct start time, grid dimensions.
- Run `real.exe` for this domain. This will produce a `wrfinput_d01` file.
- Rename this `wrfinput_d01` file to `wrfndi_d02`.

Step 3: Make the final fine-grid initial and boundary condition files

- Edit `namelist.input` again, and this time one needs to edit two columns: one for dimensions of the coarse grid, and one for the fine grid. Note that the boundary condition frequency (namelist variable `interval_seconds`) is the time in seconds between the coarse-grid model output times.
- Run `ndown.exe`, with inputs from the coarse grid `wrfout` file(s), and `wrfndi_d02` file generated from Step 2 above. This will produce `wrfinput_d02` and `wrfbdy_d02` files.

Note that program `ndown` may be run serially or in MPI, depending on the selected compile option. The `ndown` program must be built to support nesting, however. To run the program, type,

```
./ndown.exe
```
or
```
mpirun -np 4 ./ndown.exe
```

Step 4: Make the fine-grid WRF run

- Rename `wrfinput_d02` and `wrfbdy_d02` to `wrfinput_d01` and `wrfbdy_d01`, respectively.
- Edit `namelist.input` one more time, and it is now for the fine-grid domain only.
- Run WRF for this grid.

The figure on the next page summarizes the data flow for a one-way nested run using program `ndown`.

f. Moving-Nested Run

Two types of moving tests are allowed in WRF. In the first option, a user specifies the nest movement in the namelist. The second option is to move the nest automatically based on an automatic vortex-following algorithm. This option is designed to follow the movement of a well-defined tropical cyclone.

To make the specified moving nest run, select the right nesting compile option (option 'preset moves'). To run the model, only the coarse grid input files are required. In this option, the nest initialization is defined from the coarse grid data - no nest input is used. In addition to the namelist options applied to a nested run, the following needs to be added to namelist section `&domains`:

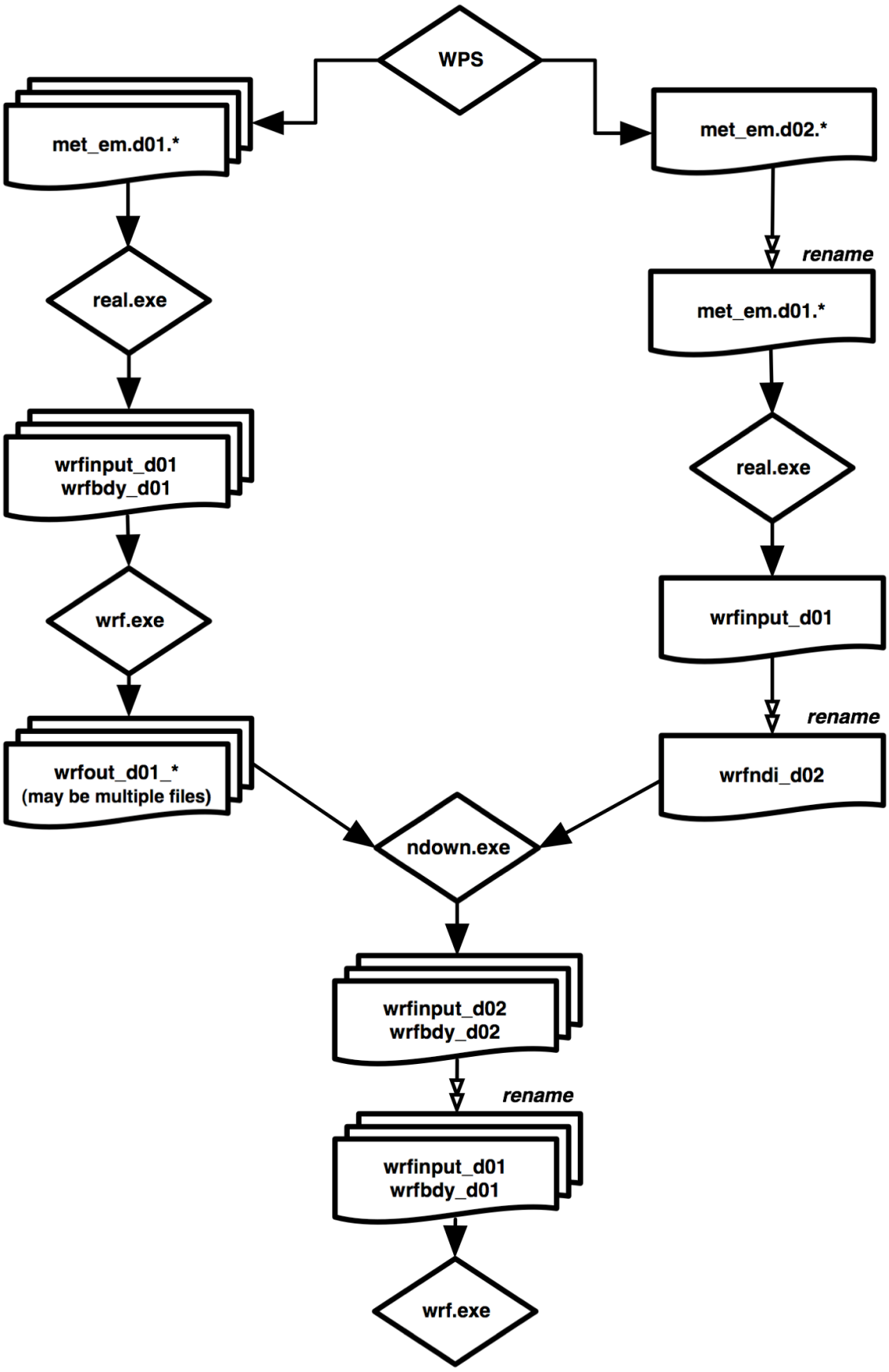
`num_moves`: the total number of moves one can make in a model run. A move of any domain counts against this total. The maximum is currently set to 50, but it can be changed by change MAX_MOVES in `frame/module_driver_constants.F`.

`move_id`: a list of nest IDs, one per move, indicating which domain is to move for a given move.

`move_interval`: the number of minutes since the beginning of the run that a move is supposed to occur. The nest will move on the next time step after the specified instant of model time has passed.

`move_cd_x`,`move_cd_y`: distance in number of grid points and direction of the nest move(positive numbers indicating moving toward east and north, while negative numbers indicating moving toward west and south).

Parameter max_moves is set to be 50, but can be modified in source code file *frame/module_driver_constants.F* if needed.


WPS
met_em.d01.*
real.exe
wrfinput_d01
wrfbdy_d01
wrf.exe
wrfout_d01_*
(may be multiple files)
met_em.d02.*
rename
met_em.d01.*
real.exe
wrfinput_d01
rename
wrfndi_d02
ndown.exe
wrfinput_d02
wrfbdy_d02
rename
wrfinput_d01
wrfbdy_d01
wrf.exe

To make the automatic moving nest runs, select the 'vortex-following' option when configuring. (***Note*** that this compile would only support auto-moving nest, and will not support the specified moving nest at the same time.) Again, no nest input is needed. If one wants to use values other than the default ones, add and edit the following namelist variables in `&domains` section:

`vortex_interval`: how often the vortex position is calculated in minutes (default is 15 minutes).

`max_vortex_speed`: used with *vortex_interval* to compute the radius of search for the new vortex center position (default is 40 m/sec).

`corral_dist`: the distance in number of coarse grid cells that the moving nest is allowed to come near the coarse grid boundary (default is 8).

`track_level`: the pressure level (in Pa) where the vortex is tracked.

In both types of moving nest runs, the initial location of the nest is specified through `i_parent_start` and `j_parent_start` in the namelist.input file.

The automatic moving nest works best for well-developed vortex.

g. Three-Dimensional Analysis Nudging Run

Prepare input data to WRF as usual using WPS. If nudging is desired in the nest domains, make sure all time periods for all domains are processed in WPS.

Set the following options before running `real.exe`, in addition to others described earlier (see namelist template `namelist.input.grid_fdda` in `test/em_real/` directory for guidance):

`grid_fdda` = *1*

Run `real.exe` as before, and this will create, in addition to `wrfinput_d0*` and `wrfbdy_d01` files, a file named '`wrffdda_d0*`'. Other grid nudging namelists are ignored at this stage. But it is a good practice to fill them all before one runs `real`. In particular, set

`gfdda_inname` = "`wrffdda_d<domain>`"
`gfdda_interval` = time interval of input data in minutes
`gfdda_end_h` = end time of grid nudging in hours

See http://www.mmm.ucar.edu/wrf/users/wrfv2/How_to_run_grid_fdda.html and `README.grid_fdda` in `WRFV3/test/em_real/` for more information.

h. Observation Nudging Run

In addition to the usual input data preparation using WPS, station observation files are required. See http://www.mmm.ucar.edu/wrf/users/wrfv2/How_to_run_obs_fdda.html for instructions. The observation file names expected by WRF are `OBS_DOMAIN101` for domain 1, and `OBS_DOMAIN201` for domain 2, etc.

Observation nudging is activated in the model by the following namelists:

`obs_nudge_opt` = *1*
`fdda_start` = *0* (obs nudging start time in minutes)
`fdda_end` = *360* (obs nudging end time in minutes)

Look for example to set other obs nudging namelist variables in namelist template `namelist.input.obs_fdda` in `test/em_real/` directory. See http://www.mmm.ucar.edu/wrf/users/wrfv2/How_to_run_obs_fdda.html and `README.obs_fdda` in `WRFV3/test/em_real/` for more information.

i. Global Run

WRFV3 begins to support global capability. To make a global run, run WPS starting with namelist template `namelist.wps.gloabl`. Set `map_proj = 'lat-lon'`. Run the rest of WPS programs as usual but only for one time period. This is because the domain covers the entire global, lateral boundary conditions are no longer needed.

Run program `real.exe` as usual but only for one time period. Lateral boundary file `wrfbdy_d01` is not needed.

Copy over `namelist.input.global` to `namelist.input`, and edit it. Run the model as usual.

Note that since this is a new option in the model, use it with caution. Not all options have been tested. For example, all filter options have not been tested.

j. Using Digital Filter Initialization

Digital filter initialization (DFI) is a way to remove initial model noise as measured by the surface pressure tendency. It runs a digital filter during a model integration, backward and forward, and then start the forecast. In WRF implementation, this can all be done in one job run. In V3, DFI can only be used in a single domain run.

No special requirement for data preparation.

Start with namelist template `namelist.input.dfi`. This namelist file contains an extra namelist record for DFI: `&dfi_control`. Edit it to match your case configuration. For a typical application, the following options are used:

`Dfi_opt` = *3*
`dfi_nfilter` = *7* (filter option: Dolph)
`dfi_cutoff_seconds` = *3600* (should not be longer than the filter window)

For time specification, it typically needs to integrate backward for 0.5 0 1 hour, and integrate forward for half of the time.

If option `dfi_write_filtered_input` is set to true, a filtered wrfinput file, `wrfinput_initialized_d01`, will be produced.

Check Output

Once a model run is completed, it is a good practice to check a couple of things quickly.

If you have run the model on multiple processors using MPI, you should have a number of `rsl.out.*` and `rsl.error.*` files. Type '`tail rsl.out.0000`' to see if you get '`SUCCESS COMPLETE WRF`'. This is a good indication that the model has run successfully.

The namelist options are written to a separate file: `namelist.output`.

Check the output times written to `wrfout*` file by using netCDF command:

```
ncdump -v Times wrfout_d01_yyyy-mm-dd_hh:00:00
```

Take a look at either `rsl.out.0000` file or other standard out file. This file logs the times taken to compute for one model time step, and to write one history and restart output:

```
Timing for main: time 2006-01-21_23:55:00 on domain  2:    4.91110 elapsed seconds.
Timing for main: time 2006-01-21_23:56:00 on domain  2:    4.73350 elapsed seconds.
Timing for main: time 2006-01-21_23:57:00 on domain  2:    4.72360 elapsed seconds.
Timing for main: time 2006-01-21_23:57:00 on domain  1:   19.55880 elapsed seconds.
```

and

```
Timing for Writing wrfout_d02_2006-01-22_00:00:00 for domain 2: 1.17970 elapsed seconds.
Timing for main: time 2006-01-22_00:00:00 on domain 1: 27.66230 elapsed seconds.
Timing for Writing wrfout_d01_2006-01-22_00:00:00 for domain 1: 0.60250 elapsed seconds.
```

If the model did not run to completion, take a look at these standard output/error files too. If the model has become numerically unstable, it may have violated the CFL criterion (for numerical stability). Check whether this is true by typing the following:

`grep cfl rsl.error.*` or `grep cfl wrf.out`

you might see something like these:

```
5 points exceeded cfl=2 in domain            1 at time   4.200000
  MAX AT i,j,k:          123          48          3 cfl,w,d(eta)= 4.165821
21 points exceeded cfl=2 in domain           1 at time   4.200000
   MAX AT i,j,k:          123          49          4 cfl,w,d(eta)= 10.66290
```

When this happens, often reducing time step can help.

Trouble Shooting

If the model aborts very quickly, it is likely that either the computer memory is not large enough to run the specific configuration, or the input data have some serious problem. For the first problem, try to type '`unlimit`' to see if more memory can be obtained.

To check if the input data is the problem, use ncview or other netCDF file browser.

Another frequent error seen is '`module_configure: initial_config: error reading namelist`'. This is an error message from the model complaining about errors and typos in the `namelist.input` file. Edit `namelist.input` file with caution. If unsure, always start with an available template. A namelist record where the namelist read error occurs is provided in the V3 error message, and it should help with identifying the error.

Physics and Dynamics Options

Physics Options

WRF offers multiple physics options that can be combined in any way. The options typically range from simple and efficient to sophisticated and more computationally costly, and from newly developed schemes to well tried schemes such as those in current operational models.

The choices vary with each major WRF release, but here we will outline those available in WRF Version 3.

1. Microphysics *(mp_physics)*

a. Kessler scheme: A warm-rain (i.e. no ice) scheme used commonly in idealized cloud modeling studies (*mp_physics* = 1).

b. Lin et al. scheme: A sophisticated scheme that has ice, snow and graupel processes, suitable for real-data high-resolution simulations (2).

c. WRF Single-Moment 3-class scheme: A simple efficient scheme with ice and snow processes suitable for mesoscale grid sizes (3).

d. WRF Single-Moment 5-class scheme: A slightly more sophisticated version of (c) that allows for mixed-phase processes and super-cooled water (4).

e. Eta microphysics: The operational microphysics in NCEP models. A simple efficient scheme with diagnostic mixed-phase processes (5).

f. WRF Single-Moment 6-class scheme: A scheme with ice, snow and graupel processes suitable for high-resolution simulations (6).

g. Goddard microphysics scheme. A scheme with ice, snow and graupel processes suitable for high-resolution simulations (7). New in Version 3.0.

h. Thompson et al. scheme: A new scheme with ice, snow and graupel processes suitable for high-resolution simulations (8; replacing the version in 2.1)

i. Morrison double-moment scheme (10). Double-moment ice, snow, rain and graupel for cloud-resolving simulations. New in Version 3.0.

2.1 Longwave Radiation *(ra_lw_physics)*

a. RRTM scheme: Rapid Radiative Transfer Model. An accurate scheme using look-up tables for efficiency. Accounts for multiple bands, trace gases, and microphysics species (*ra_lw_physics* = 1).

b. GFDL scheme: Eta operational radiation scheme. An older multi-band scheme with carbon dioxide, ozone and microphysics effects (99).

c. CAM scheme: from the CAM 3 climate model used in CCSM. Allows for aerosols and trace gases (3).

2.2 Shortwave Radiation *(ra_sw_physics)*

a. Dudhia scheme: Simple downward integration allowing efficiently for clouds and clear-sky absorption and scattering. When used in high-resolution simulations, sloping and shadowing effects may be considered (*ra_sw_physics* = 1).

b. Goddard shortwave: Two-stream multi-band scheme with ozone from climatology and cloud effects (2).

c. GFDL shortwave: Eta operational scheme. Two-stream multi-band scheme with ozone from climatology and cloud effects (99).

d. CAM scheme: from the CAM 3 climate model used in CCSM. Allows for aerosols and trace gases (3).

3.1 Surface Layer *(sf_sfclay_physics)*

a.MM5 similarity: Based on Monin-Obukhov with Carslon-Boland viscous sub-layer and standard similarity functions from look-up tables (*sf_sfclay_physics* = 1).

b. Eta similarity: Used in Eta model. Based on Monin-Obukhov with Zilitinkevich thermal roughness length and standard similarity functions from look-up tables(2).

c. Pleim-Xiu surface layer. (7). New in Version 3.0.

3.2 Land Surface *(sf_surface_physics)*

a.5-layer thermal diffusion: Soil temperature only scheme, using five layers (*sf_surface_physics* = 1).

b. Noah Land Surface Model: Unified NCEP/NCAR/AFWA scheme with soil temperature and moisture in four layers, fractional snow cover and frozen soil physics (2).

-Urban canopy model (***ucmcall***): 3-category UCM option

c. RUC Land Surface Model: RUC operational scheme with soil temperature and moisture in six layers, multi-layer snow and frozen soil physics (3).

d. Pleim-Xiu Land Surface Model. Two-layer scheme with vegetation and sub-grid tiling (7). New in Version 3.0.

4. Planetary Boundary layer *(bl_pbl_physics)*

a. Yonsei University scheme: Non-local-K scheme with explicit entrainment layer and parabolic K profile in unstable mixed layer (*bl_pbl_physics* = 1).

b. Mellor-Yamada-Janjic scheme: Eta operational scheme. One-dimensional prognostic turbulent kinetic energy scheme with local vertical mixing (2).

c. MRF scheme: Older version of (a) with implicit treatment of entrainment layer as part of non-local-K mixed layer (99).

d. ACM PBL. Asymmetric Convective Model with non-local upward mixing and local downward mixing (7). New in Version 3.0.

5. Cumulus Parameterization *(cu_physics)*

a. Kain-Fritsch scheme: Deep and shallow convection sub-grid scheme using a mass flux approach with downdrafts and CAPE removal time scale (*cu_physics* = 1).

b. Betts-Miller-Janjic scheme. Operational Eta scheme. Column moist adjustment scheme relaxing towards a well-mixed profile (2).

c. Grell-Devenyi ensemble scheme: Multi-closure, multi-parameter, ensemble method with typically 144 sub-grid members (3).

d. Grell 3d ensemble cumulus scheme. Scheme for higher resolution domains allowing for subsidence in neighboring columns (5). New in Version 3.0.

e. Old Kain-Fritsch scheme: Deep convection scheme using a mass flux approach with downdrafts and CAPE removal time scale (99).

Diffusion and Damping Options

Diffusion in WRF is categorized under two parameters, the diffusion option and the K option. The diffusion option selects how the derivatives used in diffusion are calculated, and the K option selects how the K coefficients are calculated. Note that when a PBL

option is selected, vertical diffusion is done by the PBL scheme, and not by the diffusion scheme.

1.1 Diffusion Option *(diff_opt)*

a. Simple diffusion: Gradients are simply taken along coordinate surfaces (*diff_opt* = 1).

b. Full diffusion: Gradients use full metric terms to more accurately compute horizontal gradients in sloped coordinates (*diff_opt* = 2).

1.2 K Option *(km_opt)*

Note that when using a PBL scheme, only options (a) and (d) below make sense, because (b) and (c) are designed for 3d diffusion.

a. Constant: K is specified by namelist values for horizontal and vertical diffusion (*km_opt* = 1).

b. 3d TKE: A prognostic equation for turbulent kinetic energy is used, and K is based on TKE (*km_opt* = 2).

c. 3d Deformation: K is diagnosed from 3d deformation and stability following a Smagorinsky approach (*km_opt* = 3).

d. 2d Deformation: K for horizontal diffusion is diagnosed from just horizontal deformation. The vertical diffusion is assumed to be done by the PBL scheme (*km_opt* = 4).

1.3 6th Order Horizontal Diffusion *(diff_6th_opt)*

6th-orderhorizontal hyper diffusion (del^6) on all variables to act as a selective short-wave numerical noise filter. Can be used in conjunction with diff_opt.

2. Damping Options

These are independently activated choices.

a. Upper Damping: Either a layer of increased diffusion (*damp_opt* =1) or a Rayleigh relaxation layer (2) or an implicit gravity-wave damping layer (3, new in Version 3.0), can be added near the model top to control reflection from the upper boundary.

b. w-Damping: For operational robustness, vertical motion can be damped to prevent the model from becoming unstable with locally large vertical velocities. This only affects strong updraft cores, so has very little impact on results otherwise.

c. Divergence Damping: Controls horizontally propagating sound waves.

d. External Mode Damping: Controls upper-surface (external) waves.

e. Time Off-centering (epssm): Controls vertically propagating sound waves.

Advection Options

a. Horizontal advection orders for momentum (*h_mom_adv_order*) and scalar (*h_sca_adv_order*) can be 2nd to 6th, with 5th order being the recommended one.

b. Vertical advection orders for momentum (*v_mom_adv_order*) and scalar (v_sca_adv_order) can be 2nd and 6th, with 3rd order being the recommended one.

c. Positive-definite advection option can be applied to moisture (*pd_moist*= .true.), scalar (*pd_scalar*), chemistry variables (*pd_chem*) and tke (*pd_tke*).

Other Dynamics Options

a. The model can be run hydrostatically by setting *non_hydrostatic* switch to .false.

b. Coriolis term can be applied to wind perturbation (*pert_coriolis* = .true.) only (idealized only).

c. For *diff_opt* = 2 only, vertical diffusion may act on full fields (not just on perturbation from 1D base profile (*mix_full_fields* = .true.; idealized only).

Lateral Boundary Condition Options

a. Periodic (*periodic_x* / *periodic_y*): for idealized cases.

b. Open (*open_xs, open_xe, open_ys, open_ye*): for idealized cases.

c. Symmetric (*symmetric_xs, symmetric_xe, symmetric_ys, symmetric_ye*): for idealized cases.

d. Specified (*specified*): for real-data cases. The first row and column are specified with external model values (*spec_zone* = 1, and it should not change). The rows and columns in *relax_zone* have values blended from external model and WRF. The value of *relax_zone* may be changed, as long as *spec_bdy_width* = *spec_zone* + *relax_zone*.

   spec_exp: exponential multiplier for relaxation zone ramp, used with *specified* boundary condition. 0. = linear ramp, default; 0.33 = ~3*dx exp decay factor. May be useful for long simulations.

e. Nested (*nested*): for real and idealized cases.

Description of Namelist Variables

The following is a description of namelist variables. The variables that are a function of nests are indicated by *(max_dom)* following the variable. Also see `README.namelist` file in WRFV3/run/ directory.

Variable Names	Value	Description
`&time_control`		Time control
`run_days`	1	run time in days
`run_hours`	0	run time in hours Note: if it is more than 1 day, one may use both run_days and run_hours or just run_hours. e.g. if the total run length is 36 hrs, you may set run_days = 1, and run_hours = 12, or run_days = 0, and run_hours 36
`run_minutes`	0	run time in minutes
`run_seconds`	0	run time in seconds
`start_year (max_dom)`	2001	four digit year of starting time
`start_month (max_dom)`	06	two digit month of starting time
`start_day (max_dom)`	11	two digit day of starting time
`start_hour (max_dom)`	12	two digit hour of starting time
`start_minute (max_dom)`	00	two digit minute of starting time
`start_second (max_dom)`	00	two digit second of starting time Note: the start time is used to name the first wrfout file. It also controls the start time for nest domains, and the time to restart
`end_year (max_dom)`	2001	four digit year of ending time
`end_month (max_dom)`	06	two digit month of ending time
`end_day (max_dom)`	12	two digit day of ending time
`end_hour (max_dom)`	12	two digit hour of ending time
`end_minute (max_dom)`	00	two digit minute of ending time
`end_second (max_dom)`	00	two digit second of ending time Note all end times also control when the nest domain integrations end. All start and end times are used by *real.exe*. One may use either run_days/run_hours etc. or end_year/month/day/hour etc. to control

		the length of model integration. But run_days/run_hours takes precedence over the end times. Program *real.exe* uses start and end times only.
`interval_seconds`	10800	time interval between incoming real data, which will be the interval between the lateral boundary condition file (for *real* only)
`input_from_file (max_dom)`	T (logical)	logical; whether nested run will have input files for domains other than 1
`fine_input_stream (max_dom)`		selected fields from nest input
	0	all fields from nest input are used
	2	only nest input specified from input stream 2 (defined in the Registry) are used
`history_interval (max_dom)`	60	history output file interval in minutes (integer only)
`history_interval_mo (max_dom)`	1	history output file interval in months (integer); used as alternative to history_interval
`history_interval_d (max_dom)`	1	history output file interval in days (integer); used as alternative to history_interval
`history_interval_h (max_dom)`	1	history output file interval in hours (integer); used as alternative to history_interval
`history_interval_m (max_dom)`	1	history output file interval in minutes (integer); used as alternative to history_interval and is equivalent to history_interval
`history_interval_s (max_dom)`	1	history output file interval in seconds (integer); used as alternative to history_interval
`frames_per_outfile (max_dom)`	1	output times per history output file, used to split output files into smaller pieces
`restart`	F (logical)	whether this run is a restart run
`restart_interval`	1440	restart output file interval in minutes
`reset_simulation_start`	F	whether to overwrite simulation_start_date with forecast start time
`auxinput1_inname`	“met_em.d<domain>	input from WPS (this is the default)

	<date>”	
auxinput4_inname	“wrflowinp_d<domain>”	input for lower bdy file, works with sst_update = 1
auxinput4_interval	360	file interval in minutes for lower bdy file
io_form_history	2	2 = netCDF; 102 = split netCDF files one per processor (no supported post-processing software for split files)
	1	binary format (no supported post-processing software avail)
	4	PHDF5 format (no supported post-processing software avail)
	5	GRIB 1
	10	GRIB 2
io_form_restart	2	2 = netCDF; 102 = split netCDF files one per processor (must restart with the same number of processors)
io_form_input	2	2 = netCDF
io_form_boundary	2	netCDF format
debug_level	0	50,100,200,300 values give increasing prints
auxhist2_outname	"rainfall_d<domain>"	file name for extra output; if not specified, auxhist2_d<domain>_<date> will be used. Also note that to write variables in output other than the history file requires Registry.EM file change
auxhist2_interval	10	interval in minutes
io_form_auxhist2	2	output in netCDF
frame_per_auxhist4 (max_dom)		output times per output file
auxinput11_interval		designated for obs nudging input
auxinput11_end_h		designated for obs nudging input
nocolons	.false.	replace : with _ in output file names
write_input	t	write input-formatted data as output for 3DVAR application
inputout_interval	180	interval in minutes when writing input-formatted data
input_outname	“wrf_3dvar_input_ d<domain>_<date>”	Output file name from 3DVAR
inputout_begin_y	0	beginning year to write 3DVAR data

inputout_begin_mo	0	beginning month to write 3DVAR data
inputout_begin_d	0	beginning day to write 3DVAR data
inputout_begin_h	3	beginning hour to write 3DVAR data
Inputout_begin_m	0	beginning minute to write 3DVAR data
inputout_begin_s	0	beginning second to write 3DVAR data
inputout_end_y	0	ending year to write 3DVAR data
inputout_end_mo	0	ending month to write 3DVAR data
inputout_end_d	0	ending day to write 3DVAR data
inputout_end_h	12	ending hour to write 3DVAR data
Inputout_end_m	0	ending minute to write 3DVAR data
inputout_end_s	0	ending second to write 3DVAR data. The above example shows that the input-formatted data are output starting from hour 3 to hour 12 in 180 min interval.
&domains		domain definition: dimensions, nesting parameters
time_step	60	time step for integration in integer seconds (recommended 6*dx in km for a typical case)
time_step_fract_num	0	numerator for fractional time step
time_step_fract_den	1	denominator for fractional time step Example, if you want to use 60.3 sec as your time step, set time_step = 60, time_step_fract_num = 3, and time_step_fract_den = 10
max_dom	1	number of domains - set it to > 1 if it is a nested run
s_we (max_dom)	1	start index in x (west-east) direction (leave as is)
e_we (max_dom)	91	end index in x (west-east) direction (staggered dimension)
s_sn (max_dom)	1	start index in y (south-north) direction (leave as is)
e_sn (max_dom)	82	end index in y (south-north) direction (staggered dimension)
s_vert (max_dom)	1	start index in z (vertical) direction (leave as is)

e_vert (max_dom)	28	end index in z (vertical) direction (staggered dimension - this refers to full levels). Most variables are on unstaggered levels. Vertical dimensions need to be the same for all nests.
num_metgrid_levels	40	number of vertical levels in WPS output: type ncdump -h to find out
eta_levels	1.0, 0.99,…0.0	model *eta* levels from 1 to 0. If not given, *real* will provide a set of levels
force_sfc_in_vinterp	1	use surface data as lower boundary when interpolating through this many eta levels
p_top_requested	5000	p_top to use in the model; must be available in WPS data
interp_type	1	vertical interpolation; 1: linear in pressure; 2: linear in log(pressure)
extrap_type	2	vertical extrapolation of non-temperature variables. 1: extrapolate using the two lowest levels; 2: use lowest level as constant below ground
t_extrap_type	2	vertical extrapolation for potential temperature. 1: isothermal; 2: -6.5 K/km lapse rate for temperature 3: constant theta
use_levels_below_ground	.true.	in vertical interpolation, whether to use levels below input surface level: true: use input isobaric levels below input surface false: extrapolate when WRF location is below input surface level
use_surface	.true.	whether to use input surface level data in vertical interpolation true: use input surface data false: do not use input surface data
lagrange_order	1	vertical interpolation order; 1: linear; 2: quadratic
lowest_lev_from_sfc	.false.	T = use surface values for the lowest *eta* (u,v,t,q); F = use traditional interpolation
dx (max_dom)	10000	grid length in x direction, unit in meters

dy (max_dom)	10000	grid length in y direction, unit in meters
ztop (max_dom)	19000.	height in meters; used to define model top for idealized cases
grid_id (max_dom)	1	domain identifier
parent_id (max_dom)	0	id of the parent domain
i_parent_start (max_dom)	1	starting LLC I-indices from the parent domain
j_parent_start (max_dom)	1	starting LLC J-indices from the parent domain
parent_grid_ratio (max_dom)	1	parent-to-nest domain grid size ratio: for real-data cases the ratio has to be odd; for idealized cases, the ratio can be even if feedback is set to 0.
parent_time_step_ratio (max_dom)	1	parent-to-nest time step ratio; it can be different from the parent_grid_ratio
feedback	1	feedback from nest to its parent domain; 0 = no feedback
smooth_option	0	smoothing option for parent domain, used only with feedback option on. 0: no smoothing; 1: 1-2-1 smoothing; 2: smoothing-desmoothing

(options for preset moving nest)

num_moves	2,	total number of moves for all domains
move_id (max_moves)	2,2,	a list of nest domain id's, one per move
move_interval (max_moves)	60,120,	time in minutes since the start of this domain
move_cd_x (max_moves)	1,-1,	the number of parent domain grid cells to move in i direction
move_cd_y (max_moves)	-1,1,	the number of parent domain grid cells to move in j direction (positive in increasing i/j directions, and negative in decreasing i/j directions. Only 1, 0 and -1 is permitted.

(options for automatic moving nest)

vortex_interval (max_dom)	15	how often the new vortex position is computed
max_vortex_speed (max_dom)	40	unit in m/sec; used to compute the search radius for the new vortex position
corral_dist (max_dom)	8	how many coarse grid cells the moving nest is allowed to get near the coarse grid

		boundary
(options for adaptive time step)		
`use_adaptive_time_step`	.false.	whether to use adaptive time step
`step_to_output_time`	.true.	whether to modify the time steps so that the exact history time is reached
`target_cfl`	1.2	if vertical and horizontal CFL <= this value, then time step is increased
`max_step_increase_pct`	5	percentage of previous time step to increase, if the max CFL is <= `target_cfl`
`starting_time_step`	-1	flag -1 implies 6*dx is used to start the model. Any positive integer number specifies the time step the model will start with. Note that when `use_adaptive_time_step` is true, the value specified for `time_step` is ignored.
`max_time_step`	-1	flag -1 implies the maximum time step is 3*starting_time_step. Any positive integer number specified the maximum time step
`min_time_step`	-1	flag -1 implies the minimum time step is 0.5*starting_time_step. Any positive integer number specified the minumum time step
(options to control parallel computing)		
`tile_sz_x`	0	number of points in tile x direction
`tile_sz_y`	0	number of points in tile y direction can be determined automatically
`numtiles`	1	number of tiles per patch (alternative to above two items)
`nproc_x`	-1	number of processors in x for decomposition
`nproc_y`	-1	number of processors in y for decomposition -1: code will do automatic decomposition >1: for both: will be used for decomposition
`&physics`		Physics options
`mp_physics (max_dom)`		*microphysics option*

	0	no microphysics
	1	Kessler scheme
	2	Lin et al. scheme
	3	WSM 3-class simple ice scheme
	4	WSM 5-class scheme
	5	Ferrier (new Eta) microphysics
	6	WSM 6-class graupel scheme
	7	Goddard GCE scheme (also use `gsfcgce_hail` and `gsfcgce_2ice`)
	8	Thompson graupel scheme
	10	Morrison 2-moment scheme
`mp_zero_out`		For non-zero `mp_physics` options, this keeps moisture variables above a threshold value >= 0.
	0	no action taken, no adjustment to any moisture field
	1	except for Qv, all other moisture arrays are set to zero if they fall below a critical value
	2	Qv >= 0 and all other moisture arrays are set to zero if they fall below a critical value
`mp_zero_out_thresh`	1.e-8	critical value for moisture variable threshold, below which moisture arrays (except for Qv) are set to zero (unit: kg/kg)
`gsfcgce_hail`	0	0: running gsfcgce scheme with graupel 1: running gsfcgce scheme with hail
`gsfcgce_2ice`	0	0: running gsfcgce scheme with snow, ice and graupel / hail 1: running gsfcgce scheme with only ice and snow 2: running gsfcgce scheme with only ice and graupel (used only in very extreme situation)
`no_mp_heating`	0	switch to turn off latent heating from mp 0: normal 1: turn off latent heating from a microphysics scheme
`ra_lw_physics (max_dom)`		longwave radiation option

	0	no longwave radiation
	1	rrtm scheme
	3	CAM scheme
	99	GFDL (Eta) longwave (semi-supported)
ra_sw_physics (max_dom)		shortwave radiation option
	0	no shortwave radiation
	1	Dudhia scheme
	2	Goddard short wave
	3	CAM scheme
	99	GFDL (Eta) longwave (semi-supported)
radt (max_dom)	30	minutes between radiation physics calls. Recommend 1 minute per km of dx (e.g. 10 for 10 km grid); use the same value for all nests
co2tf	1	CO2 transmission function flag for GFDL radiation only. Set it to 1 for ARW, which allows generation of CO2 function internally
cam_abs_freq_s	21600	CAM clear sky longwave absorption calculation frequency (recommended minimum value to speed scheme up)
levsiz	59	for CAM radiation input ozone levels
paerlev	29	for CAM radiation input aerosol levels
cam_abs_dim1	4	for CAM absorption save array
cam_abs_dim2	same as e_vert	for CAM 2nd absorption save array
sf_sfclay_physics (max_dom)		surface-layer option
	0	no surface-layer
	1	Monin-Obukhov scheme
	2	Monin-Obukhov (Janjic Eta) scheme
	3	NCEP GFS scheme (NMM only)
	7	Pleim-Xu (ARW only), only tested with Pleim-Xu surface and ACM2 PBL
sf_surface_physics (max_dom)		land-surface option (set before running *real*; also set correct num_soil_layers)
	0	no surface temp prediction
	1	thermal diffusion scheme

	2	unified Noah land-surface model
	3	RUC land-surface model
	7	Pleim-Xu scheme (ARW only)
bl_pbl_physics (max_dom)		boundary-layer option
	0	no boundary-layer
	1	YSU scheme
	2	Mellor-Yamada-Janjic (Eta) TKE scheme
	3	NCEP GFS scheme (NMM only)
	7	ACM2 (Pleim) scheme
	99	MRF scheme (to be removed)
bldt (max_dom)	0	minutes between boundary-layer physics calls. 0 = call every time step
cu_physics (max_dom)		cumulus option
	0	no cumulus
	1	Kain-Fritsch (new Eta) scheme
	2	Betts-Miller-Janjic scheme
	3	Grell-Devenyi ensemble scheme
	4	Simplied Arakawa-Schubert (NMM only)
	5	New Grell scheme (G3)
	99	previous Kain-Fritsch scheme
cudt	0	minutes between cumulus physics calls. 0 = call every time step
isfflx	1	heat and moisture fluxes from the surface (only works for sf_sfclay_physics = 1) 1 = with fluxes from the surface 0 = no flux from the surface
ifsnow	0	snow-cover effects (only works for sf_surface_physics = 1) 1 = with snow-cover effect 0 = without snow-cover effect
icloud	1	cloud effect to the optical depth in radiation (only works for ra_sw_physics = 1 and ra_lw_physics = 1) 1 = with cloud effect 0 = without cloud effect
swrat_scat	1.	Scattering tuning parameter (default 1 is 1.e-5 m^2/kg)
surface_input_source	1,2	where landuse and soil category data

		come from: 1 = WPS/geogrid; 2 = GRIB data from another model (only if arrays VEGCAT/SOILCAT exist)
num_soil_layers		number of soil layers in land surface model (set in *real*)
	5	thermal diffusion scheme for temp only
	4	Noah land-surface model
	6	RUC land-surface model
	2	Pleim-Xu land-surface model
pxlsm_smois_init (max_dom)	1	PX LSM soil moisture initialization option 0: from analysis 1: from LANDUSE.TBL (SLMO)
ucmcall (max_dom)	0	activate urban canopy model (in Noah LSM only) (0=no, 1=yes)
maxiens	1	Grell-Devenyi only
maxens	3	G-D only
maxens2	3	G-D only
maxens3	16	G-D only
ensdim	144	G-D only. These are recommended numbers. If you would like to use any other number, consult the code, know what you are doing.
seaice_threshold	271.	tsk < seaice_threshold, if water point and 5-layer slab scheme, set to land point and permanent ice; if water point and Noah scheme, set to land point, permanent ice, set temps from 3 m to surface, and set smois and sh2o
sst_update		option to use time-varying SST, seaice, vegetation fraction, and albedo during a model simulation (set before running *real*)
	0	no SST update
	1	*real.exe* will create `wrflowinp_d01` file at the same time interval as the available input data. To use it in wrf.exe, add `auxinput4_inname = "wrflowinp_d<domain>"`, `auxinput4_interval` in namelist section *&time_control*
usemonalb	.false.	whether to use monthly albedo map

		instead of LANDUSE.TBL values. Recommended for `sst_update = 1`
`slope_rad`	0	slope effects for `ra_sw_physics=1` (1=on, 0=off)
`topo_shading`	0	neighboring-point shadow effects for `ra_sw_physics=1` (1=on, 0=off)
`shadlen`	25000.	max shadow length in meters for `topo_shading = 1`
`omlcall`	0	simple ocean mixed layer model. (1=on, 0=off)
`oml_hml0`	50.	initial ocean mixed layer depth (m), constant everywhere
`oml_gamma`	0.14	lapse rate in deep water for oml (K m-1)
`isftcflx`	0	alternative Ck, Cd for tropical storm application. (1=on, 0=off)

`&fdda` *for grid and obs nudging*

(for grid nudging)

`grid_fdda (max_dom)`	1	grid-nudging on (=0 off) for each domain
`gfdda_inname`	"wrffdda_d<domain>"	Defined name in real
`gfdda_interval (max_dom)`	360	Time interval (min) between analysis times
`gfdda_end_h (max_dom)`	6	Time (h) to stop nudging after start of forecast
`io_form_gfdda`	2	Analysis format (2 = netcdf)
`fgdt (max_dom)`	0	Calculation frequency (in minutes) for analysis nudging. 0 = every time step, and this is recommended
`if_no_pbl_nudging_uv (max_dom)`	0	1= no nudging of u and v in the pbl; 0= nudging in the pbl
`if_no_pbl_nudging_t (max_dom)`	0	1= no nudging of temp in the pbl; 0= nudging in the pbl
`if_no_pbl_nudging_t (max_dom)`	0	1= no nudging of qvapor in the pbl; 0= nudging in the pbl
`if_zfac_uv (max_dom)`	0	0= nudge u and v all layers, 1= limit nudging to levels above k_zfac_uv
`k_zfac_uv`	10	10=model level below which nudging is switched off for u and v
`if_zfac_t (max_dom)`	0	

k_zfac_t	10	10=model level below which nudging is switched off for temp
if_zfac_q (max_dom)	0	
k_zfac_q	10	10=model level below which nudging is switched off for water qvapor
guv (max_dom)	0.0003	nudging coefficient for u and v (sec-1)
gt (max_dom)	0.0003	nudging coefficient for temp (sec-1)
gq (max_dom)	0.0003	nudging coefficient for qvapor (sec-1)
if_ramping	0	0= nudging ends as a step function, 1= ramping nudging down at end of period
dtramp_min	60.	time (min) for ramping function, 60.0=ramping starts at last analysis time, -60.0=ramping ends at last analysis time
(for obs nudging)		
obs_nudge_opt (max_dom)	1	obs-nudging fdda on (=0 off) for each domain; also need to set auxinput11_interval and auxinput11_end_h in time_control namelist
max_obs	150000	max number of observations used on a domain during any given time window
fdda_start	0.	obs nudging start time in minutes
fdda_end	180.	obs nudging end time in minutes
obs_nudge_wind (max_dom)	1	whether to nudge wind: (=0 off)
obs_coef_wind (max_dom)	6.e-4	nudging coefficient for wind, unit: s-1
obs_nudge_temp (max_dom)	1	whether to nudge temperature: (=0 off)
obs_coef_temp (max_dom)	6.e-4	nudging coefficient for temp, unit: s-1
obs_nudge_mois (max_dom)	1	whether to nudge water vapor mixing ratio: (=0 off)
obs_coef_mois (max_dom)	6.e-4	nudging coefficient for water vapor mixing ratio, unit: s-1
obs_nudge_pstr (max_dom)	0	whether to nudge surface pressure (not used)
obs_coef_pstr (max_dom)	0.	nudging coefficient for surface pressure, unit: s-1 (not used)
obs_rinxy	200.	horizontal radius of influence in km

obs_rinsig	0.1	vertical radius of influence in *eta*
obs_twindo (max_dom)	0.666667	half-period time window over which an observation will be used for nudging; the unit is in hours
obs_npfi	10	freq in coarse grid timesteps for diag prints
obs_ionf (max_dom)	2	freq in coarse grid timesteps for obs input and err calc
obs_idynin	0	for dynamic initialization using a ramp-down function to gradually turn off the FDDA before the pure forecast (=1 on)
obs_dtramp	40.	time period in minutes over which the nudging is ramped down from one to zero.
obs_nobs_prt (max_dom)	10	number of current obs to print grid coord. info.
obs_ipf_in4dob	.true.	print obs input diagnostics (=.false. off)
obs_ipf_errob	.true.	print obs error diagnostics (=.false. off)
obs_ipf_nudob	.true.	print obs nudge diagnostics (=.false. off)
obs_ipf_init	.true.	enable obs init warning messages
&dynamics		*Diffusion, damping options, advection options*
rk_ord		time-integration scheme option:
	2	Runge-Kutta 2nd order
	3	Runge-Kutta 3rd order (recommended)
diff_opt		turbulence and mixing option:
	0	= no turbulence or explicit spatial numerical filters (km_opt IS IGNORED).
	1	evaluates 2nd order diffusion term on coordinate surfaces. uses kvdif for vertical diff unless PBL option is used. may be used with km_opt = 1 and 4. (= 1, recommended for real-data case)
	2	evaluates mixing terms in physical space (stress form) (x,y,z). turbulence parameterization is chosen by specifying km_opt.
km_opt		eddy coefficient option
	1	constant (use khdif and kvdif)

	2	1.5 order TKE closure (3D)
	3	Smagorinsky first order closure (3D) Note: option 2 and 3 are not recommended for DX > 2 km
	4	horizontal Smagorinsky first order closure (recommended for real-data case)
`diff_6th_opt (max_dom)`	0	6th-order numerical diffusion 0 = no 6th-order diffusion (default) 1 = 6th-order numerical diffusion 2 = 6th-order numerical diffusion but prohibit up-gradient diffusion
`diff_6th_factor (max_dom)`	0.12	6th-order numerical diffusion non-dimensional rate (max value 1.0 corresponds to complete removal of 2dx wave in one timestep)
`damp_opt`		upper level damping flag
	0	without damping
	1	with diffusive damping; maybe used for real-data cases (`dampcoef` nondimensional ~ 0.01 - 0.1)
	2	with Rayleigh damping (`dampcoef` inverse time scale [1/s], e.g. 0.003)
	3	with w-Rayleigh damping (`dampcoef` inverse time scale [1/s] e.g. .05; for real-data cases)
`zdamp (max_dom)`	5000	damping depth (m) from model top
`dampcoef (max_dom)`	0.	damping coefficient (see `damp_opt`)
`w_damping`		vertical velocity damping flag (for operational use)
	0	without damping
	1	with damping
`base_pres`	100000.	Base state surface pressure (Pa), real only. Do not change.
`base_temp`	290.	Base state sea level temperature (K), real only.
`base_lapse`	50.	real-data ONLY, lapse rate (K), DO NOT CHANGE.
`khdif (max_dom)`	0	horizontal diffusion constant (m^2/s)
`kvdif (max_dom)`	0	vertical diffusion constant (m^2/s)

smdiv (max_dom)	0.1	divergence damping (0.1 is typical)
emdiv (max_dom)	0.01	external-mode filter coef for mass coordinate model (0.01 is typical for real-data cases)
epssm (max_dom)	.1	time off-centering for vertical sound waves
non_hydrostatic (max_dom)	.true.	whether running the model in hydrostatic or non-hydro mode
pert_coriolis (max_dom)	.false.	Coriolis only acts on wind perturbation (idealized)
top_lid (max_dom)	.false.	zero vertical motion at top of domain
mix_full_fields	.false.	used with diff_opt = 2; value of ".true." is recommended, except for highly idealized numerical tests; damp_opt must not be 1 if ".true." is chosen. .false. means subtract 1-d base-state profile before mixing
mix_isotropic(max_dom)	0	0=anistropic vertical/horizontal diffusion coeffs, 1=isotropic
mix_upper_bound(max_dom)	0.1	non-dimensional upper limit for diffusion coeffs
h_mom_adv_order (max_dom)	5	horizontal momentum advection order (5=5th, etc.)
v_mom_adv_order (max_dom)	3	vertical momentum advection order
h_sca_adv_order (max_dom)	5	horizontal scalar advection order
v_sca_adv_order (max_dom)	3	vertical scalar advection order
time_step_sound (max_dom)	4	number of sound steps per time-step (if using a time_step much larger than 6*dx (in km), increase number of sound steps). = 0: the value computed automatically
pd_moist (max_dom)	.false.	positive define advection of moisture; set to .true. to turn it on
pd_scalar (max_dom)	.false.	positive define advection of scalars
pd_tke (max_dom)	.false.	positive define advection of tke
pd_chem (max_dom)	.false.	positive define advection of chem vars
tke_drag_coefficient (max_dom)	0	surface drag coefficient (Cd, dimensionless) for diff_opt=2 only
tke_heat_flux (max_dom)	0	surface thermal flux (H/rho*cp), K m/s) for diff_opt = 2 only

do_coriolis (max_dom)	.true.	whether to do Coriolis calculations (idealized)
do_curvature (max_dom)	.true.	whether to do curvature calculations (idealized)
do_gradp (max_dom)	.true.	whether to do horizontal pressure gradient calculations (idealized)
fft_filter_lat	45.	the latitude above which the polar filter is turned on for global model
&bdy_control		*boundary condition control*
spec_bdy_width	5	total number of rows for specified boundary value nudging
spec_zone	1	number of points in specified zone (spec b.c. option)
relax_zone	4	number of points in relaxation zone (spec b.c. option)
specified (max_dom)	.false.	specified boundary conditions (only can be used for to domain 1)
spec_exp	0.	exponential multiplier for relaxation zone ramp for specified=.t. (0.= linear ramp default; 0.33=~3*dx exp decay factor)
		The above 5 namelists are used for real-data runs only
periodic_x (max_dom)	.false.	periodic boundary conditions in x direction
symmetric_xs (max_dom)	.false.	symmetric boundary conditions at x start (west)
symmetric_xe (max_dom)	.false.	symmetric boundary conditions at x end (east)
open_xs (max_dom)	.false.	open boundary conditions at x start (west)
open_xe (max_dom)	.false.	open boundary conditions at x end (east)
periodic_y (max_dom)	.false.	periodic boundary conditions in y direction
symmetric_ys (max_dom)	.false.	symmetric boundary conditions at y start (south)
symmetric_ye (max_dom)	.false.	symmetric boundary conditions at y end (north)
open_ys (max_dom)	.false.	open boundary conditions at y start (south)

open_ye (max_dom)	.false.	open boundary conditions at y end (north)
nested (max_dom)	.false.,.true.,.true.,	nested boundary conditions (must be set to .true. for nests)
polar	.false.	polar boundary condition (v=0 at polarward-most v-point) for global application
&namelist_quilt		*Option for asynchronized I/O for MPI applications*
nio_tasks_per_group	0	default value is 0: no quilting; > 0 quilting I/O
nio_groups	1	default 1
&grib2		
background_proc_id	255	Background generating process identifier, typically defined by the originating center to identify the background data that was used in creating the data. This is octet 13 of Section 4 in the grib2 message
forecast_proc_id	255	Analysis or generating forecast process identifier, typically defined by the originating center to identify the forecast process that was used to generate the data. This is octet 14 of Section 4 in the grib2 message
production_status	255	Production status of processed data in the grib2 message. See Code Table 1.3 of the grib2 manual. This is octet 20 of Section 1 in the grib2 record
compression	40	The compression method to encode the output grib2 message. Only 40 for jpeg2000 or 41 for PNG are supported
&dfi_control	digital filter option control (does not yet support nesting)	
dfi_opt	3	which DFI option to use 0: no digital filter initialization 1: digital filter launch (DFL) 2: diabatic DFI (DDFI) 3: twice DFI (TDFI) (recommended)
dfi_nfilter	7	digital filter type: 0 – uniform; 1-Lanczos; 2 – Hamming; 3 – Blackman; 4

		– Kaiser; 5 – Potter; 6 – Dolph window; 7 – Dolph (recommended); 8 – recursive high-order
dfi_write_filtered_input	.true.	whether to write wrfinput file with filtered model state before beginning forecast
dfi_write_dfi_history	.false.	whether to write wrfout files during filtering integration
dfi_cutoff_seconds	3600	cutoff period, in seconds, for the filter. Should not be longer than the filter window
dfi_time_dim	1000	maximum number of time steps for filtering period, this value can be larger than necessary
dfi_bckstop_year	2001	four-digit year of stop time for backward DFI integration. For a model that starts from 2001061112, this specifies 1 hour backward integration
dfi_bckstop_month	06	two-digit month of stop time for backward DFI integration
dfi_bckstop_day	11	two-digit day of stop time for backward DFI integration
dfi_bckstop_hour	11	two-digit hour of stop time for backward DFI integration
dfi_bckstop_minute	00	two-digit minute of stop time for backward DFI integration
dfi_bckstop_second	00	two-digit second of stop time for backward DFI integration
dfi_fwdstop_year	2001	four-digit year of stop time for forward DFI integration. For a model that starts at 2001061112, this specifies 30 minutes of forward integration
dfi_fwdstop_month	06	two-digit month of stop time for forward DFI integration
dfi_fwdstop_day	11	two-digit day of stop time for forward DFI integration
dfi_fwdstop_hour	12	two-digit hour of stop time for forward DFI integration
dfi_fwdstop_minute	30	two-digit minute of stop time for forward DFI integration
dfi_fwdstop_second	00	two-digit second of stop time for forward DFI integration

List of Fields in WRF Output

List of Fields

The following is an edited output from netCDF command *'ncdump'*. Note that valid output fields will depend on the model options used.

```
ncdump -h wrfout_d01_yyyy_mm_dd-hh:mm:ss
```

```
 netcdf wrfout_d01_2000-01-24_12:00:00 {
dimensions:
        Time= UNLIMITED ; // (1 currently)
        DateStrLen= 19 ;
        west_east= 73 ;
        south_north= 60 ;
        west_east_stag= 74 ;
        bottom_top= 27 ;
        south_north_stag= 61 ;
        bottom_top_stag= 28 ;
        soil_layers_stag= 5 ;
variables:
        charTimes(Time, DateStrLen) ;
        floatLU_INDEX(Time, south_north, west_east) ;
                LU_INDEX:description= "LAND USE CATEGORY" ;
                LU_INDEX:units= "" ;
        floatU(Time, bottom_top, south_north, west_east_stag) ;
                U:description= "x-wind component" ;
                U:units= "m s-1" ;
        floatV(Time, bottom_top, south_north_stag, west_east) ;
                V:description= "y-wind component" ;
                V:units= "m s-1" ;
        floatW(Time, bottom_top_stag, south_north, west_east) ;
                W:description= "z-wind component" ;
                W:units= "m s-1" ;
        floatPH(Time, bottom_top_stag, south_north, west_east) ;
                PH:description= "perturbation geopotential" ;
                PH:units= "m2 s-2" ;
        floatPHB(Time, bottom_top_stag, south_north, west_east) ;
                PHB:description= "base-state geopotential" ;
                PHB:units= "m2 s-2" ;
        floatT(Time, bottom_top, south_north, west_east) ;
                T:description= "perturbation potential temperature(theta-t0)" ;
                T:units= "K" ;
        floatMU(Time, south_north, west_east) ;
                MU:description= "perturbation dry air mass in column" ;
                MU:units= "Pa" ;
        floatMUB(Time, south_north, west_east) ;
                MUB:description= "base state dry air mass in column" ;
                MUB:units= "Pa" ;
        floatNEST_POS(Time, south_north, west_east) ;
                NEST_POS:description= "-" ;
                NEST_POS:units= "-" ;
        floatP(Time, bottom_top, south_north, west_east) ;
                P:description= "perturbation pressure" ;
                P:units= "Pa" ;
        floatPB(Time, bottom_top, south_north, west_east) ;
                PB:description= "BASE STATE PRESSURE" ;
                PB:units= "Pa" ;
        floatSR(Time, south_north, west_east) ;
                SR:description= "fraction of frozen precipitation" ;
                SR:units= "-" ;
        floatFNM(Time, bottom_top) ;
                FNM:description= "upper weight for vertical stretching" ;
```

```
                FNM:units= "" ;
        floatFNP(Time, bottom_top) ;
                FNP:description= "lower weight for vertical stretching" ;
                FNP:units= "" ;
        floatRDNW(Time, bottom_top) ;
                RDNW:description= "inverse d(eta) values between full (w)
levels" ;
                RDNW:units= "" ;
        floatRDN(Time, bottom_top) ;
                RDN:description= "inverse d(eta) values between half (mass)
levels" ;
                RDN:units= "" ;
        floatDNW(Time, bottom_top) ;
                DNW:description= "d(eta) values between full (w) levels" ;
                DNW:units= "" ;
        floatDN(Time, bottom_top) ;
                DN:description= "d(eta) values between half (mass) levels" ;
                DN:units= "" ;
        floatZNU(Time, bottom_top) ;
                ZNU:description= "eta values on half (mass) levels" ;
                ZNU:units= "" ;
        floatZNW(Time, bottom_top_stag) ;
                ZNW:description= "eta values on full (w) levels" ;
                ZNW:units= "" ;
        floatCFN(Time) ;
                CFN:description= "extrapolation constant" ;
                CFN:units= "" ;
        floatCFN1(Time) ;
                CFN1:description= "extrapolation constant" ;
                CFN1:units= "" ;
        floatQ2(Time, south_north, west_east) ;
                Q2:description= "QV at 2 M" ;
                Q2:units= "kg kg-1" ;
        floatT2(Time, south_north, west_east) ;
                T2:description= "TEMP at 2 M" ;
                T2:units= "K" ;
        floatTH2(Time, south_north, west_east) ;
                TH2:description= "POT TEMP at 2 M" ;
                TH2:units= "K" ;
        floatPSFC(Time, south_north, west_east) ;
                PSFC:description= "SFC PRESSURE" ;
                PSFC:units= "Pa" ;
        floatU10(Time, south_north, west_east) ;
                U10:description= "U at 10 M" ;
                U10:units= "m s-1" ;
        floatV10(Time, south_north, west_east) ;
                V10:description= "V at 10 M" ;
                V10:units= "m s-1" ;
        floatRDX(Time) ;
                RDX:description= "INVERSE X GRID LENGTH" ;
                RDX:units= "" ;
        floatRDY(Time) ;
                RDY:description= "INVERSE Y GRID LENGTH" ;
                RDY:units= "" ;
        floatRESM(Time) ;
                RESM:description= "TIME WEIGHT CONSTANT FOR SMALL STEPS" ;
                RESM:units= "" ;
        floatZETATOP(Time) ;
                ZETATOP:description= "ZETA AT MODEL TOP" ;
                ZETATOP:units= "" ;
        floatCF1(Time) ;
                CF1:description= "2nd order extrapolation constant" ;
                CF1:units= "" ;
        floatCF2(Time) ;
                CF2:description= "2nd order extrapolation constant" ;
```

```
        CF2:units= "" ;
floatCF3(Time) ;
        CF3:description= "2nd order extrapolation constant" ;
        CF3:units= "" ;
intITIMESTEP(Time) ;
        ITIMESTEP:description= "" ;
        ITIMESTEP:units= "" ;
floatXTIME(Time) ;
        XTIME:description= "minutes since simulation start" ;
        XTIME:units= "" ;
floatQVAPOR(Time, bottom_top, south_north, west_east) ;
        QVAPOR:description= "Water vapor mixing ratio" ;
        QVAPOR:units= "kg kg-1" ;
floatQCLOUD(Time, bottom_top, south_north, west_east) ;
        QCLOUD:description= "Cloud water mixing ratio" ;
        QCLOUD:units= "kg kg-1" ;
floatQRAIN(Time, bottom_top, south_north, west_east) ;
        QRAIN:description= "Rain water mixing ratio" ;
        QRAIN:units= "kg kg-1" ;
floatLANDMASK(Time, south_north, west_east) ;
        LANDMASK:description= "LAND MASK (1 FOR LAND, 0 FOR WATER)" ;
        LANDMASK:units= "" ;
floatTSLB(Time, soil_layers_stag, south_north, west_east) ;
        TSLB:description= "SOIL TEMPERATURE" ;
        TSLB:units= "K" ;
floatZS(Time, soil_layers_stag) ;
        ZS:description= "DEPTHS OF CENTERS OF SOIL LAYERS" ;
        ZS:units= "m" ;
floatDZS(Time, soil_layers_stag) ;
        DZS:description= "THICKNESSES OF SOIL LAYERS" ;
        DZS:units= "m" ;
floatSMOIS(Time, soil_layers_stag, south_north, west_east) ;
         SMOIS:description= "SOIL MOISTURE" ;
        SMOIS:units= "m3 m-3" ;
floatSH2O(Time, soil_layers_stag, south_north, west_east) ;
        SH2O:description= "SOIL LIQUID WATER" ;
        SH2O:units= "m3 m-3" ;
floatXICE(Time, south_north, west_east) ;
        XICE:description= "SEA ICE FLAG" ;
        XICE:units= "" ;
floatSFROFF(Time, south_north, west_east) ;
        SFROFF:description= "SURFACE RUNOFF" ;
        SFROFF:units= "mm" ;
floatUDROFF(Time, south_north, west_east) ;
        UDROFF:description= "UNDERGROUND RUNOFF" ;
        UDROFF:units= "mm" ;
intIVGTYP(Time, south_north, west_east) ;
        IVGTYP:description= "DOMINANT VEGETATION CATEGORY" ;
        IVGTYP:units= "" ;
intISLTYP(Time, south_north, west_east) ;
        ISLTYP:description= "DOMINANT SOIL CATEGORY" ;
        ISLTYP:units= "" ;
floatVEGFRA(Time, south_north, west_east) ;
        VEGFRA:description= "VEGETATION FRACTION" ;
        VEGFRA:units= "" ;
floatGRDFLX(Time, south_north, west_east) ;
        GRDFLX:description= "GROUND HEAT FLUX" ;
        GRDFLX:units= "W m-2" ;
floatSNOW(Time, south_north, west_east) ;
        SNOW:description= "SNOW WATER EQUIVALENT" ;
        SNOW:units= "kg m-2" ;
floatSNOWH(Time, south_north, west_east) ;
        SNOWH:description= "PHYSICAL SNOW DEPTH" ;
        SNOWH:units= "m" ;
floatRHOSN(Time, south_north, west_east) ;
```

```
                RHOSN:description= " SNOW DENSITY" ;
                RHOSN:units= "kg m-3" ;
        floatCANWAT(Time, south_north, west_east) ;
                CANWAT:description= "CANOPY WATER" ;
                CANWAT:units= "kg m-2" ;
        floatSST(Time, south_north, west_east) ;
                SST:description= "SEA SURFACE TEMPERATURE" ;
                SST:units= "K" ;
        floatQNDROPSOURCE(Time, bottom_top, south_north, west_east) ;
                QNDROPSOURCE:description= "Droplet number source" ;
                QNDROPSOURCE:units= " /kg/s" ;
        floatMAPFAC_M(Time, south_north, west_east) ;
                MAPFAC_M:description= "Map scale factor on mass grid" ;
                MAPFAC_M:units= "" ;
        floatMAPFAC_U(Time, south_north, west_east_stag) ;
                MAPFAC_U:description= "Map scale factor on u-grid" ;
                MAPFAC_U:units= "" ;
        floatMAPFAC_V(Time, south_north_stag, west_east) ;
                MAPFAC_V:description= "Map scale factor on v-grid" ;
                MAPFAC_V:units= "" ;
        floatF(Time, south_north, west_east) ;
                F:description= "Coriolis sine latitude term" ;
                F:units= "s-1" ;
        floatE(Time, south_north, west_east) ;
                E:description= "Coriolis cosine latitude term" ;
                E:units= "s-1" ;
        floatSINALPHA(Time, south_north, west_east) ;
                SINALPHA:description= "Local sine of map rotation" ;
                SINALPHA:units= "" ;
        floatCOSALPHA(Time, south_north, west_east) ;
                COSALPHA:description= "Local cosine of map rotation" ;
                COSALPHA:units= "" ;
        floatHGT(Time, south_north, west_east) ;
                HGT:description= "Terrain Height" ;
                HGT:units= "m" ;
        floatTSK(Time, south_north, west_east) ;
                TSK:description= "SURFACE SKIN TEMPERATURE" ;
                TSK:units= "K" ;
        floatP_TOP(Time) ;
                P_TOP:description= "PRESSURE TOP OF THE MODEL" ;
                P_TOP:units= "Pa" ;
        floatRAINC(Time, south_north, west_east) ;
                RAINC:description= "ACCUMULATED TOTAL CUMULUS PRECIPITATION" ;
                RAINC:units= "mm" ;
        floatRAINNC(Time, south_north, west_east) ;
                RAINNC:description= "ACCUMULATED TOTAL GRID SCALE
PRECIPITATION" ;
                RAINNC:units= "mm" ;
        floatSNOWNC(Time, south_north, west_east) ;
                SNOWNC:description="ACCUMULATED TOTAL GRIDSCALE SNOW AND ICE" ;
                SNOWNC:units= "mm" ;
        floatGRAUPELNC(Time, south_north, west_east) ;
                GRAUPELNC:description= "ACCUMULATED TOTAL GRID SCALE GRAUPEL" ;
                GRAUPELNC:units= "mm" ;
        floatSWDOWN(Time, south_north, west_east) ;
                SWDOWN:description= "DOWNWARD SHORT WAVE FLUX AT GROUND
SURFACE" ;
                SWDOWN:units= "W m-2" ;
        floatGLW(Time, south_north, west_east) ;
                GLW:description= "DOWNWARD LONG WAVE FLUX AT GROUND SURFACE" ;
                GLW:units= "W m-2" ;
        floatOLR(Time, south_north, west_east) ;
                OLR:description= "TOA OUTGOING LONG WAVE" ;
                OLR:units= "W m-2" ;
        floatXLAT(Time, south_north, west_east) ;
```

```
                XLAT:description= "LATITUDE, SOUTH IS NEGATIVE" ;
                XLAT:units= "degree_north" ;
        floatXLONG(Time, south_north, west_east) ;
                XLONG:description= "LONGITUDE, WEST IS NEGATIVE" ;
                XLONG:units= "degree_east" ;
        floatXLAT_U(Time, south_north, west_east_stag) ;
                XLAT_U:description= "LATITUDE, SOUTH IS NEGATIVE" ;
                XLAT_U:units= "degree_north" ;
        floatXLONG_U(Time, south_north, west_east_stag) ;
                XLONG_U:description= "LONGITUDE, WEST IS NEGATIVE" ;
                XLONG_U:units= "degree_east" ;
        floatXLAT_V(Time, south_north_stag, west_east) ;
                XLAT_V:description= "LATITUDE, SOUTH IS NEGATIVE" ;
                XLAT_V:units= "degree_north" ;
        floatXLONG_V(Time, south_north_stag, west_east) ;
                XLONG_V:description= "LONGITUDE, WEST IS NEGATIVE" ;
                XLONG_V:units= "degree_east" ;
        floatALBEDO(Time, south_north, west_east) ;
                ALBEDO:description= "ALBEDO" ;
                ALBEDO:units= "-" ;
        floatTMN(Time, south_north, west_east) ;
                TMN:description= "SOIL TEMPERATURE AT LOWER BOUNDARY" ;
                TMN:units= "K" ;
        floatXLAND(Time, south_north, west_east) ;
                XLAND:description= "LAND MASK (1 FOR LAND, 2 FOR WATER)" ;
                XLAND:units= "" ;
        floatUST(Time, south_north, west_east) ;
                UST:description= "U* IN SIMILARITY THEORY" ;
                UST:units= "m s-1" ;
        floatPBLH(Time, south_north, west_east) ;
                PBLH:description= "PBL HEIGHT" ;
                PBLH:units= "m" ;
        floatHFX(Time, south_north, west_east) ;
                HFX:description= "UPWARD HEAT FLUX AT THE SURFACE" ;
                HFX:units= "W m-2" ;
        floatQFX(Time, south_north, west_east) ;
                QFX:description= "UPWARD MOISTURE FLUX AT THE SURFACE" ;
                QFX:units= "kg m-2 s-1" ;
        floatLH(Time, south_north, west_east) ;
                LH:description= "LATENT HEAT FLUX AT THE SURFACE" ;
                LH:units= "W m-2" ;
        floatSNOWC(Time, south_north, west_east) ;
                SNOWC:description= "FLAG INDICATING SNOW COVERAGE (1 FOR SNOW
COVER)" ;
                SNOWC:units= "" ;
}
```

Special WRF Output Variables

WRF model outputs the state variables defined in the Registry file, and these state variables are used in the model's prognostic equations. Some of these variables are perturbation fields. Therefore some definition for reconstructing meteorological variables is necessary. In particular, the definitions for the following variables are:

total geopotential	`PH + PHB`
total geopotential height in m	`( PH + PHB ) / 9.81`

total potential temperature in_ K	`T + 300`
total pressure in mb	`( P + PB ) * 0.01`

The definition for map projection options:

map_proj = 1: Lambert Conformal
map_proj = 2: Polar Stereographic
map_proj = 3: Mercator
map_proj = 10: latitude and longitude

List of Global Attributes

```
// global attributes:
                :TITLE= " OUTPUT FROM WRF V3.0 MODEL" ;
                :START_DATE= "2000-01-24_12:00:00" ;
                :SIMULATION_START_DATE= "2000-01-24_12:00:00" ;
                :WEST-EAST_GRID_DIMENSION= 74 ;
                :SOUTH-NORTH_GRID_DIMENSION= 61 ;
                :BOTTOM-TOP_GRID_DIMENSION= 28 ;
                :DX= 30000.f ;
                :DY= 30000.f ;
                :GRIDTYPE= "C" ;
                :DYN_OPT= 2 ;
                :DIFF_OPT= 1 ;
                :KM_OPT= 4 ;
                :DAMP_OPT= 0 ;
                :KHDIF= 0.f ;
                :KVDIF= 0.f ;
                :MP_PHYSICS= 3 ;
                :RA_LW_PHYSICS= 0 ;
                :RA_SW_PHYSICS= 1 ;
                :SF_SFCLAY_PHYSICS= 1 ;
                :SF_SURFACE_PHYSICS= 1 ;
                :BL_PBL_PHYSICS= 1 ;
                :CU_PHYSICS= 1 ;
                :SURFACE_INPUT_SOURCE= 1 ;
                :SST_UPDATE= 0 ;
                :GRID_FDDA= 0 ;
                :GFDDA_INTERVAL_M= 0 ;
                :GFDDA_END_H= 0 ;
                :UCMCALL= 0 ;
                :FEEDBACK= 1 ;
                :SMOOTH_OPTION= 0 ;
                :SWRAD_SCAT= 1.f ;
                :W_DAMPING= 0 ;
                :PD_MOIST= 1 ;
                :PD_SCALAR= 0 ;
                :PD_TKE= 0 ;
                :DIFF_6TH_OPT= 0 ;
                :DIFF_6TH_FACTOR= 0.12f ;
                :OBS_NUDGE_OPT= 0 ;
                :WEST-EAST_PATCH_START_UNSTAG= 1 ;
                :WEST-EAST_PATCH_END_UNSTAG= 73 ;
                :WEST-EAST_PATCH_START_STAG= 1 ;
                :WEST-EAST_PATCH_END_STAG= 74 ;
                :SOUTH-NORTH_PATCH_START_UNSTAG= 1 ;
                :SOUTH-NORTH_PATCH_END_UNSTAG= 60 ;
                :SOUTH-NORTH_PATCH_START_STAG= 1 ;
                :SOUTH-NORTH_PATCH_END_STAG= 61 ;
                :BOTTOM-TOP_PATCH_START_UNSTAG= 1 ;
                :BOTTOM-TOP_PATCH_END_UNSTAG= 27 ;
                :BOTTOM-TOP_PATCH_START_STAG= 1 ;
                :BOTTOM-TOP_PATCH_END_STAG= 28 ;
                :GRID_ID= 1 ;
                :PARENT_ID= 0 ;
                :I_PARENT_START= 0 ;
                :J_PARENT_START= 0 ;
                :PARENT_GRID_RATIO= 1 ;
                :DT= 180.f ;
                :CEN_LAT= 34.83001f ;
                :CEN_LON= -81.03f ;
                :TRUELAT1= 30.f ;
```

```
:TRUELAT2= 60.f ;
:MOAD_CEN_LAT= 34.83001f ;
:STAND_LON= -98.f ;
:GMT= 12.f ;
:JULYR= 2000 ;
:JULDAY= 24 ;
:MAP_PROJ= 1 ;
:MMINLU= "USGS" ;
:ISWATER= 16 ;
:ISICE= 24 ;
:ISURBAN= 1 ;
:ISOILWATER= 14 ;
```

Chapter 6: WRF-Var

Table of Contents

Introduction

Data assimilation is the technique by which **observations** are combined with an NWP product (the **first guess** or background forecast) and their respective error statistics to provide an improved estimate (the **analysis**) of the atmospheric (or oceanic, Jovian, whatever) state. Variational (Var) data assimilation achieves this through the iterative minimization of a prescribed cost (or penalty) function. Differences between the analysis and observations/first guess are penalized (damped) according to their perceived error. The difference between three-dimensional (3D-Var) and four-dimensional (4D-Var) data assimilation is the use of a numerical forecast model in the latter.

MMM Division of NCAR supports a unified (global/regional, multi-model, 3/4D-Var) model-space variational data assimilation system (WRF-Var) for use by NCAR staff and collaborators, and is also freely available to the general community, together with further documentation, test results, plans etc., from the WRF-Var web-page http://www.wrf-model.org/development/group/WG4. The documentation you are reading is the "Users Guide" for those interested in downloading and running the code. This text also forms the documentation for the online tutorial. The online WRF-Var tutorial is recommended for people who are

- Potential users of WRF-Var who want to learn how to run WRF-Var by themselves;

- New users who plan on coming to the NCAR WRF-Var tutorial - for you we recommend that you try this tutorial before you come to NCAR whether you are able or unable to register for practice sessions, and this will hopefully help you to understand the lectures a lot better;

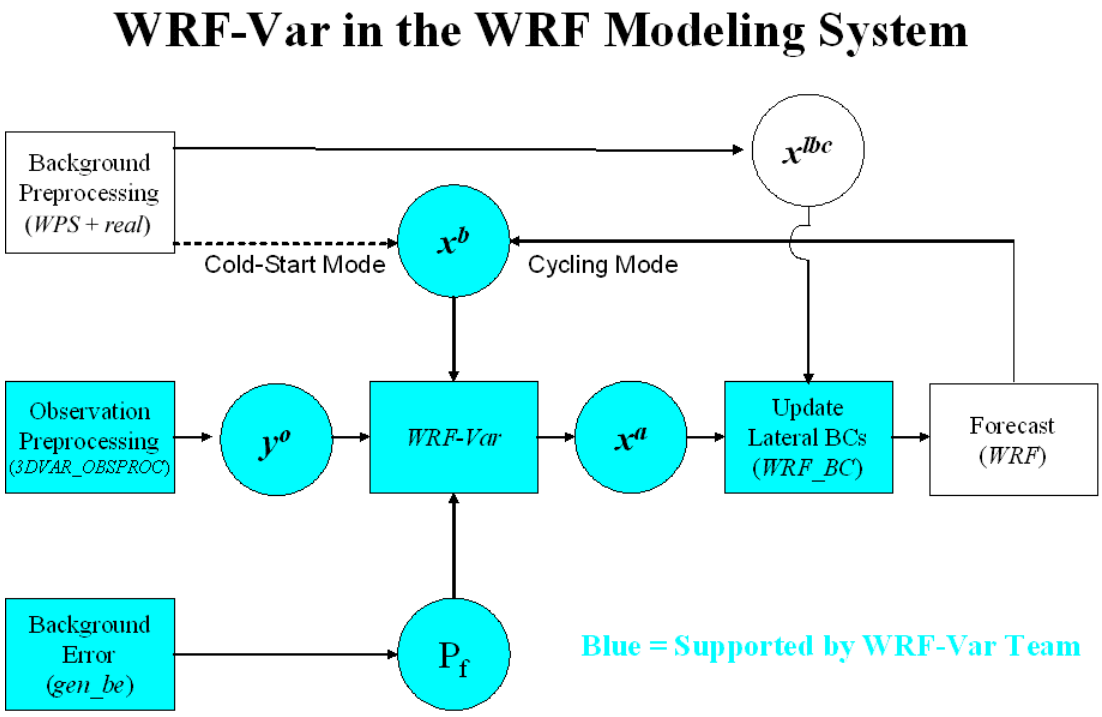
- Users who are looking for references to diagnostics, namelist options etc - look for 'Miscellanies' and 'Trouble Shooting' sections on each page.

If you are a new WRF-Var user, this tutorial is designed to take you through WRF-Var-related programs step by step. If you are familiar with 3/4D-Var systems, you may find useful information here too as the WRF-Var implementation of 3/4D-Var contains a number of unique capabilities (e.g. multiple background error models, WRF-framework based parallelism/IO, direct radar reflectivity assimilation). If you do not know anything about WRF-Var, you should first read the WRF-Var tutorial presentations available from the WRF WRF-Var web page http://www.wrf-model.org/development/group/WG4.

Goals of this WRF-Var Tutorial

In this WRF-Var tutorial, you will learn how to run the various components of WRF-Var system. In the online tutorial, you are supplied with a test case including the following input data: a) observation file, b) WRF NETCDF background file (previous forecast used as a first guess of the analysis), and c) Background error statistics (climatological estimate of errors in the background file). In your own work, you will need to create these input files yourselves.

Various components of the WRF-Var system are shown in blue in the sketch below, together with their relationship with rest of the WRF system.



Before using your own data, we suggest that you start by running through the WRF-Var related programs at least once using the supplied test case. This serves two purposes: First, you can learn how to run the programs with data we have tested ourselves, and second you can test whether your computer is adequate to run the entire modeling system. After you have done this tutorial, you can try

- Running other, more computationally intensive, case studies.
- Experimenting with some of the many namelist variables.

  WARNING: It is impossible to test every code upgrade with every permutation of computer, compiler, number of processors, case, namelist option, etc. The "namelist" options that are supported are indicated in the "Registry.wrfvar" and these are the default options. WRF-Var may be run with options other than the default option by specifying its value via the "wrapper" script. Sample of "wrapper" scripts are included in "var/scripts/wrappers" directory. For running "WRF-Var", you may like to adopt a suitable "wrapper" script in this directory and modify it depending on your case.

- Running with your own domain. Hopefully, our test cases will have prepared you (and us!) for the variety of ways in which you may wish to run WRF-Var. Please let us know your experiences.

As a professional courtesy, we request that you include the following reference in any publications that makes use of any component of the community WRF-Var system:

Barker, D.M., W. Huang, Y. R. Guo, and Q. N. Xiao., 2004: A Three-Dimensional (3DVAR)Data Assimilation System For Use With MM5: Implementation and Initial Results. *Mon. Wea. Rev.*, **132**, 897-914.

As you are going through the online tutorial, you will download program tar files and data to your local computer, compile and run on it. Do you know what machine you are going to use to run WRF-Var related programs? What compilers do you have on the machine?

Running WRF-Var requires a Fortran 90 compiler. We currently support the following platforms: **IBM**, **DEC, SGI**, **PC/Linux** (with Portland Group compiler), Cray-X1, and Apple G4/G5. Please let us know if this does not meet your requirements, and we will attempt to add other machines to our list of supported architectures as resources allow. Although we are interested to hear of your experiences modifying compile options, we do not yet recommend making changes to the configure file used to compile WRF-Var.

Tutorial Schedule

We recommend you follow the online tutorial in the order of the sections listed below. This tutorial does not cover parts of the larger WRF system, required if you wish to go beyond the test case supplied here, e.g. the WRF Pre-processing System (WPS) and *real* pre-preprocessors are needed to create your own background field.

The online tutorial is broken down into the following sections.

a) Download Test Data: This page describes how to access test data sets to run WRF-Var.
b) Download Source Code: This page describes how to access source code.
c) WRF-Var observation pre-processor (obsproc): This page describes how to create an observation file for subsequent use in WRF-Var, and plot observation distributions.
d) Setting up WRF-Var: In this part of the tutorial you will compile the codes that form the WRF-Var system.
e) Single Observation Test for WRF-Var In this part of the tutorial you will compile the codes that form the WRF-Var system.
f) Run WRF-Var for con200 Case Study: In this section, you will learn how to run WRF-Var for a test case.
g) WRF-Var Diagnostics: WRF-Var produces a number of diagnostics file that contain useful information on how the assimilation has performed. This section will introduce you to some of these files, and what to look for.

h) Updating WRF lateral boundary conditions: Before using WRF-Var analysis as the initial conditions for a WRF forecast, the lateral boundary file must be modified to take account of the differences between first guess and analysis.

i) Additional WRF-Var exercises: In this section you will learn more about WRF-Var, like how to run single observations test, response of background error scaling & variance, minimization aspect and how to suppress a particular type of data in WRF-Var etc

Download Test Data

This page describes how to access test datasets required to run WRF-Var. If you do not know anything about WRF-Var, you should first read the Introduction section.

Required Data

The WRF-Var system requires three input files to run: a) A WRF *first guess* input format file output from either WPS (cold-start) or WRF itself (warm-start), b) Observations (in BUFR or ASCII little_r format), and c) A background error statistics file (containing background error covariance currently calculated via the *NMC-method*), using "gen_be" utility of WRF-Var. The use of these three data files in WRF-Var is described later in the tutorial.

The following table summarizes the above info:

Input Data	*Format*	*Created By*
First Guess	NETCDF	WRF Preprocessing System (WPS) or WRF
Observations	ASCII (PREPBUFR also possible)	Observation Preprocessor (OBSPROC)
Background Error Statistics	Binary	WRF-Var *gen_be* utility

In the online tutorial, example input files are given. In your own work after the tutorial, you will need to create these input data sets yourselves.

Downloading Test Data

In the online tutorial, you will store data in a directory defined by the environment variable $*DAT_DIR*. This directory can be at any location and it should have read access. To run this case it is desired to have at least 100MB of memory to be made available on your machine (maybe much more for other cases). Type

setenv DAT_DIR dat_dir

Here, "dat_dir" is your own chosen directory where you aim to store the WRF-Var input data. Create this directory, if it does not exist, and type

cd $DAT_DIR

Download the tutorial test data for a "con200" case which is valid 00 UTC 02 January 2007 test case from

http://www.mmm.ucar.edu/wrf/src/data/WRFV3-Var-testdata.tar.gz

Once you have downloaded "WRFV3-Var-testdata.tar.gz" file to $DAT_DIR, then extract it by typing

gunzip WRFV3-Var-testdata.tar.gz

tar -xvf WRFV3-Var-testdata.tar

You should then see a number of data files that will be used in the tutorial. To save space type

rm -rf WRFV3-Var-testdata.tar

What next?

Note: You may find the following three sub-directories/files under "$DAT_DIR"

ob /2007010200/ob.little_r ### Observation data in "little_r" format

rc /2007010200/wrfinput_d01 & wrfbdy_d01 ### First guess & lateral boundary file

be/be.dat ### Background error file

Download Source code

http://www.mmm.ucar.edu/wrf/src/data/WRFV3-Var.tar.gz

Once you have downloaded source code file "WRFV3-Var.tar.gz", next step is to extract the source code by typing

gunzip WRFV3-Var.tar.gz

tar -xvf WRFV3-Var.tar

You should see a sub-directory "var". Place this under your WRFV3 directory (*which your previously downloaded*) The subdirectory "var" is the main sub-directory holding source code for both WRF-Var and "obsproc", the observation pre-processor for WRF-Var observation data input. To save the space execute following command.

rm -rf WRFV3-Var.tar

WRF-Var Observation Preprocessor (OBSPROC)

Having downloaded successfully the test data and the source code, first step is to process observation input for WRF-Var by compiling and running the obs pre-processor (OBSPROC).

Compiling OBSPROC code

WRF-Var observation preprocessor is residing under "var/obsproc" directory

cd var/obsproc

Read instructions in "README" file to compile OBSPROC and proceed by typing

make

Once this is complete (a minute or less on most machines), you can check for the presence of the OBSPROC executable by issuing following command (from "var/obsproc" directory)

ls -l src/3dvar_obs.exe

Running OBSPROC:

OK, so now you have compiled "OBSPROC". Before running "3dvar_obs.exe", create the desired namelist file ***namelist.3dvar_obs*** (see README.namelist for details);

For your reference in "var/obsproc" directory a file named "namelist_3dvar_obs.wrfvar-tut" have already been created. Thus, proceed as follows.

cp namelist.3dvar_obs.wrfvar-tut namelist.3dvar_obs

edit ***namelist.3dvar_obs***

In this namelist file, all you need is to change the full path and name of the observation file (**ob.little_r**) depending upon where this file is downloaded and where it finally resides.

To run OBSPROC, type

3dvar_obs.exe >&! 3dvar_obs.out

Looking at WRF-Var OBSPROC output.

Once ***3dvar_obs.exe*** has completed successfully, you will see an observation data file: ***obs_gts.3dvar***, in "obsproc" directory. This is the input observation file to WRF-Var. Before running WRF-Var, you may like to learn more about various types of data you are aiming to assimilate for this case, its geographical distribution etc. This file is in ASCII format and so you can easily view it. To have a graphical view about the content of this file, there is a "MAP_plot" utility to look at the data distribution for each type of observations. To exercise this, proceed as follows.

1) **cd MAP_plot**;

2) Modify the script ***Map.csh*** to set the time window and full path of input observation file (obs_gts.3dvar). Precisely, the following string in this script as follows.

TIME_WINDOW_MIN = '2007010121'

TIME_ANALYSIS = '2007010200'

TIME_WINDOW_MAX = '2007010203'

OBSDATA = ./obs_gts.3dvar

3) Type

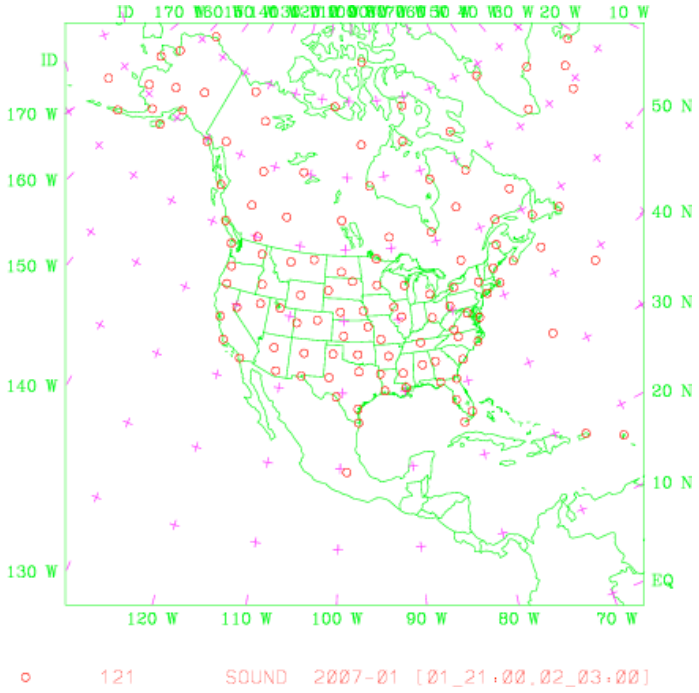
Map.csh

Note: the *configure.user* will be automatically picked up for your computer system when you type ***Map.csh***;

4) When the job has completed, you will have a gmeta file ***gmeta.****{analysis_time}* corresponding to *analysis_time*=2007010200. This contains plots of data distribution for each type of observations contained in the OBS data file: ***obs_gts.3dvar***. To view this, type

idt gmeta.2007010200

It will display (panel by panel) geographical distribution of various types of data which are listed in the header of "obs_gts.3dvar". Following is the geographic distribution of "sonde" observations for this case.



Saving necessary file for WRF-Var and clean OBSPROC

In this tutorial, we are storing data in a directory defined by the environment variable $*DAT_DIR*. After creating successfully, your own observation file **(obs_gts.3dvar),** copy it to $*DAT_DIR* using the command (from "obsproc" directory)

mv obs_gts.3dvar $*DAT_DIR*/ob/2007010200/ob.ascii

Finally, to clean up the "var/obsproc" directory, type following command in the same directory

make clean

Miscellanies:

1) When you run 3dvar_obs.exe, and you did not obtain the file *obs_gts.3dvar,* please check 3dvar_obs.out file to see where the program aborted. Usually there is information in this file to tell you what is wrong;
2) When you run 3dvar_obs.exe and got an error as *'Error in NAMELIST record 2'* in 3dvar_obs.out file, please check if your *namelist.3dvar_obs* file matches with the *Makefile* settings. Either your *namelist.3dvar_obs* file or *Makefile* need to be modified, then re-compile and re-run the job;
3) From the *.diag files, you may find which observation report caused the failure of job.
4) In most cases, the job failure was caused by incorrect input file names, or the specified analysis time, time window, etc. in innamelist.3dvar_obs;
5) If users still cannot figure out the troubles, please inform us and pass us your input files including the namelist file, and printed file 3dvar_obs.out.

What next?

OK, you have now created the observation file and looked at some plots of observations, now you are ready to move on to setting up WRF-Var.

Setting up WRF-Var

In this part of the tutorial you will compile WRF-Var code.

Compile WRF-Var code

Proceed as follows while in the main WRFV3 directory:

```
setenv WRF_DA_CORE 1

source  var/build/setup.csh

./configure
```

You will then see a list of options, depending on whether you want to run single processor, with different compilers etc. Choose proper option suiting your requirements.

In this tutorial session, you will be running WRF-Var in single processor mode. Therefore, enter **1** (single-threaded) at the prompt. This will automatically create a file *configure.wrf* in the main WRFV3 directory. Check it with the following command:

```
ls  -l  configure.wrf
```

We recommend you to run WRF-Var in single processor mode first, but later if you want you can run WRF-Var on distributed memory machines by recompiling it appropriately.

For compilation of WRF-Var code type (from the main WRFV3 directory)

./compile all_wrfvar

Successful compilation of 'all_wrfvar" will produce several executables in "var/da" directory including "da_wrfvar.exe". You can list these executables by issuing the command (from the main WRFV3 directory)

ls -l var/da/*exe

After successful compilation of WRF-Var you are ready to run WRF-Var for the test case. The WRF-Var system is run through a suitable wrapper script, which activates various standard scripts residing in "WRFV3/var/scripts" directory.

Thus to run any case like the tutorial test case in this tutorial exercise, user is supposed to write a suitable wrapper script and execute this script. For example, the wrapper script for running "con200", the tutorial test case is located at:

WRFV3/var/scripts/wrappers/da_run_suite_wrapper_con200.ksh

You need to modify this wrapper script for various job details defined via different environment variables in this wrapper script. The user should also be able to run WRF-Var with namelist options other than its pre-set default values, which are defined in WRFV3/Registry/Registry.wrfvar file. It is done by suitably defining the desired environment variables appropriate for your own application. Examples include changing data directory, source code location, experiment run area, east-west south-north, vertical dimension of your domain etc. All non-default namelist options, which user wanted to set via wrapper script will show up in the WRF-Var namelist (namelist.input) file, which is created in the run directory ($RUN_DIR) defined via the wrapper script.

Note: As a rule any WRF-Var namelist option should always be set in wrapper script using upper case letters preceded by "NL" string. For example for "con200" case the grid size dimensions in x-direction is 200000 m and the corresponding WRF-Var namelist variable name is "dx". This is specified in wrapper script as "export NL_DX=200000".

WRF-Var system uses WRF framework to define and perform parallel, I/O functions. This is fairly transparent in the WRF-Var code. At run time, WRF-Var requires one to specify the grid dimensions at run-time. This is communicated to WRF-Var system via the wrapper script like for this case as follows.

export NL_E_WE=45

export NL_E_SN=45

export NL_E_VERT=28\

Thus user needs to change these parameters to run their own case.

Note: If this grid dimensions do not mach with the dimensions written in the first guess (FG) input file, WRF-Var will abort with proper diagnostic message.

Having compiled WRF-Var and familiarized yourself with the setting of various environment variables in wrapper script, it's time to run your test case.

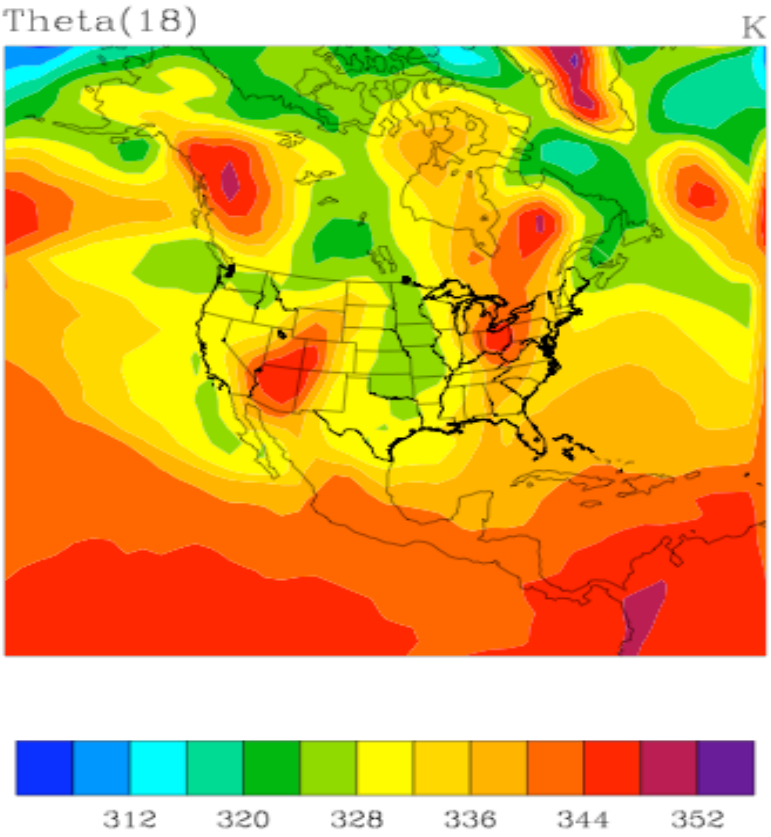
Run WRF-Var con200 case Study

In this section, you will learn how to run WRF-Var using observations and a first guess from a low-resolution (200 km) CONUS domain (see below).

By this stage you have successfully created the three input files (first guess, observation and background error statistics files in directory *$DAT_DIR*) required to run WRF-Var. Also, you have successfully downloaded and compiled the WRF-Var code. If this is correct, we are ready to learn how to run WRF-Var.

The Case

The data for this case is valid at 00 UTC 2nd January 2007. The first guess comes from the NCEP global final analysis system (FNL), passed through the WRF-WPS and *real* packages. The first-guess level 18 potential temperature field is shown below, illustrating the domain:



The intention of running this test-case is to provide a simplified, computationally cheap application in order to train potential uses of WRF-Var (this case uses only a small fraction of WRF-Var capabilities).

Note: In WRF-Var various NCL scripts are included under var/graphics/ncl sub-directory to display results.

Changes required in wrapper script to run con200 case:

Following environment variables needs to be set in

WRFV3/var/scripts/wrappers/da_run_suite_wrapper_con200.ksh script depending upon your case.

REL_DIR	- Full path of the parent code directory
WRFVAR_DIR	- Full directory path of wrfvar under $REL_DIR
DAT_DIR	- Full path of data input holding ob, be, rc directories

REGION - String representing region under $DAT_DIR

OB_DIR - Full path for Observation directory

RC_DIR - Full path for "FG" directory

BE_DIR - Full path for "background error (BE)" directory

INITIAL_DATE - Initial date for running WRF-Var

FINAL_DATE - Final date for running WRF-Var

Following environment variables are optional with default values shown in bracket:

EXPT - Experiment ID (expt)

RUN_DIR - Full path for run directory ($DAT_DIR/$REGION/expt/test_$NUM_PROC)

FC_DIR - Analysis output directory ($DAT_DIR/$REGION/expt/fc)

Note: Since output is written in $RUN_DIR, user has to ensure that this directory/area has proper write permission.

Run WRF-Var

Once you have set the necessary environment variables, in "da_run_suite_wrapper_con200.ksh" script, you can run this case by typing

da_run_suite_wrapper_con200.ksh

in WRFV3/var/scripts/wrappers subdirectory or from the directory where your modified wrapper script is located.

Successful completion of job results in a number of output diagnostic files in ${RUN_DIR} directory. The final analysis file (wrfinput_d01) will appear under $FC_DIR/2007010200 directory. Various textual diagnostics output files will be explained in the next section (WRF-Var Diagnostics). Here, we merely wish to run WRF-Var for this case.

In order to understand the role of various important WRF-Var options, try re-running WRF-Var by changing different namelist options via wrapper script. For example, making WRF-Var convergence criteria more stringent by reducing the value of "EPS" to e.g. 0.0001 by setting "NL_EPS=0.0001" in your wrapper script. If WRF-Var has not converged by the maximum number of iterations (NTMAX=200) then you may need to increase the value of NTMAX variable by setting "export NL_NTMAX=500" in your

wrapper script and may like to run the case again this case. Last section of this tutorial deals with many such WRF-Var additional exercises.

Note: You may like to change "RUN_DIR" environment variable to store results separately for each run. In your wrapper script by setting the option "CLEAN=true", you can save lot of space as this option removes contents of "working" directory.

v. What next?

Having run WRF-Var, you should now spend time looking at some of the diagnostic output files created by WRF-Var.

WRF-Var Diagnostics

WRF-Var produces a number of diagnostic files that contain useful information on how the data assimilation has performed. This section will introduce you to some of these files, and what to look for.

Which are the important diagnostic files to look for?

Having run WRF-Var, it is important to check a number of output files to see if the assimilation appears sensible. Change directory to where you ran this case:

cd ${RUN_DIR}/2007010200/wrfvar

ls -l

You will see something like the following:

```
total 10880
-rw-r--r-- 1 rizvi  ncar    280 Mar 18 15:41 cost_fn
-rw-r--r-- 1 rizvi  ncar    247 Mar 18 15:41 grad_fn
-rw-r--r-- 1 rizvi  ncar 9048641 Mar 18 15:41 gts_omb_oma
-rw-r--r-- 1 rizvi  ncar   4156 Mar 18 15:41 index.html
-rw-r--r-- 1 rizvi  ncar   1942 Mar 18 15:41 namelist.input
-rw-r--r-- 1 rizvi  ncar 76986 Mar 18 15:41 namelist.output
drwxr-xr-x 2 rizvi  ncar  65536 Mar 18 15:41 rsl
-rw-r--r-- 1 rizvi  ncar  22730 Mar 18 15:41 statistics
drwxr-xr-x 2 rizvi  ncar  65536 Mar 18 15:41 trace
drwxr-xr-x 3 rizvi  ncar  65536 Mar 18 15:41 working
```

Content of some useful diagnostic files are as follows:

cost_fn & grad_fn: These files hold (in ASCII format) WRF-Var cost & gradient function values respectively for the first and last iterations. However, if you run with "NL_CALCULATE_CG_COST_FN=true", it lists these values for each iterations. For

visualization purpose, sometimes it is good to run WRF-Var with this option, as it will list the cost and gradient function for each iteration. Following NCL script may be used to plot the content of "*cont_fn*" & "*grad_fn*", if these files are generated with "NL_CALCULATE_CG_COST_FN=true".

"WRFV3/var/graphcs/ncl/plot_cost_grad_fn.ncl"

Note: Make sure that you removed first two records (header) in *"cost_fn" & "grad_fn"*. Also you need to specify the directory name for these two files.

gts_omb_oma: It holds (in ASCII format) information of each type of observations, like its value, quality, observation error, observation minus background (OMB) & observation minus analysis (OMA). This information is very useful for (both analysis or forecasts) verification purposes.

namelist.input: WRF-Var input namelist file. It displays all the non-default options which user defined. If any of your namelist defined options did not appear in this file, you may like to check its name and match it with the "WRFV3/Registry/Registry.wrfvar".

namelist.output: Consolidated list of all the namelist options used.

rsl: Directory containing information of standard WRF-Var output from individual processors. It contains host of information on number of observations, minimization, timings etc. Additional diagnostics may be printed in these files by including various "print" WRF-Var namelist options. TO learn more about these additional "print" options, search "print_" string in "WRFV3/Registry/Registry.wrfvar".

statistics: Text file containing OMB (OI), OMA (OA) statistics (minimum, maximum, mean and standard deviation) for each observation type and variable. This information is very useful in diagnosing how WRF-Var has used different components of the observing system. Also contained are the analysis minus background (A-B) statistics i.e. statistics of the analysis increments for each model variable at each model level. This information is very useful in checking the range of analysis increment values found in the analysis, and where they are in the WRF-model grid space.

Where is the final analysis file?

The final analysis file resides as "$FC_DIR/2007010200/wrfinput_d01". It is in WRF (NETCDF) format. It me be visualized using following NCL script.

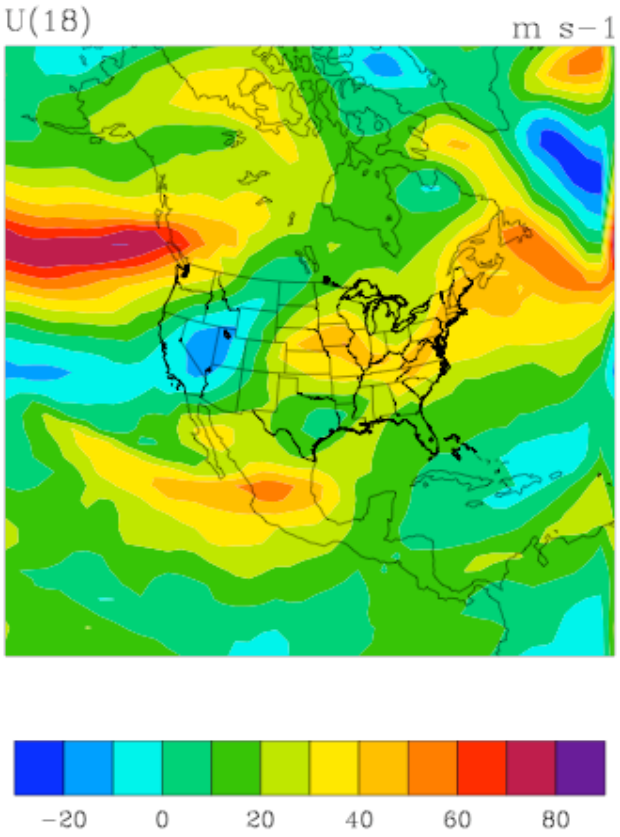
"WRFV3/var/graphcs/ncl/WRF-Var_plot.ncl"

You need to specify the *analsyis_file* name, its full path etc. The details are listed in this script as comments.

As an example, if you are aiming to display U-component of the analysis at level 18, execute following command after modifying the scrip "WRFV3/var/graphcs/ncl/WRF-Var_plot.ncl"

ncl WRF-Var_plot.ncl

It will display like:



You may like to change the variable name, level etc in this script to display the variable of you're your choice at the desired sigma level.

Take time to look through the textual output files to ensure you understand how WRF-Var has performed. For example,

How closely has WRF-Var fitted individual observation types? . Look at the *statistics* file to compare the O-B and O-A statistics.

How big are the analysis increments? Again, look in the *statistics* file to see minimum/maximum values of A-B for each variable at various levels. It will give you a

feel of the impact of input observation data you assimilated via WRF-Var by modifying the input analysis first guess.

How long did WRF-Var take to converge? Does it really converge? You will get the answers of all these questions by looking into rsl-files, as it indicates the number of iterations taken by WRF-Var to converge. If this is the same as the maximum number of iterations specified in the namelist (NTMAX) or its default value (=200) set in WRFV3/Registry/Registry.wrfvar", then it means that the analysis solution did not converge. If so, so you may like to increase the value of "NTAMAX" and rerun your case to ensure that the convergence is achieved.

Plotting WRF-Var analysis increments

A good visual way of seeing the impact of assimilation of observations is to plot the analysis increments (i.e. analysis minus first guess difference). There are many different graphics packages used (e.g. RIP, NCL, GRADS etc) that can do this. The plot of level 18 theta increments below was produced using the particular NCL script. This script is located at

"WRFV3/var/graphcs/ncl/WRF-Var_plot.ncl"

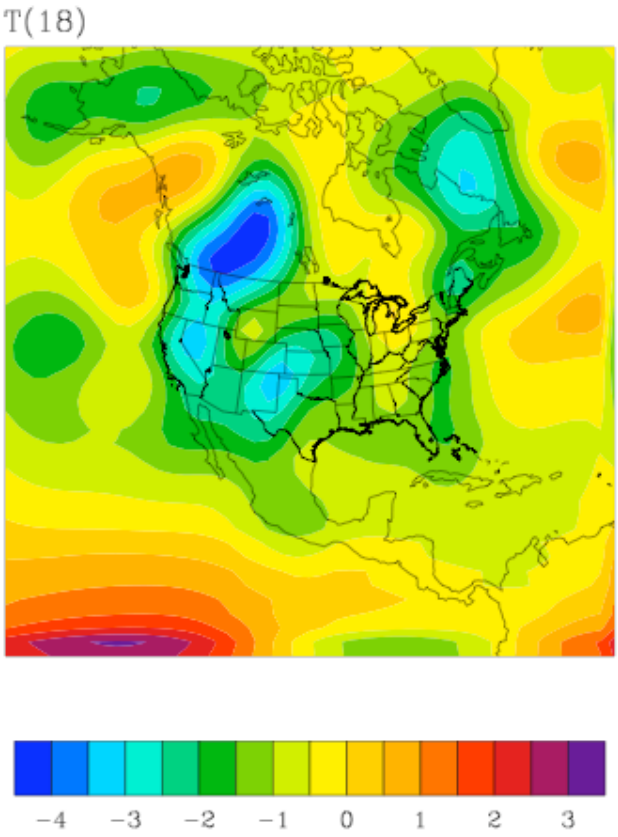
You need to modify this script to fix the full path for *first_guess & analysis* files. You may also like to modify the display level by setting "kl" and the name of the variable to display by setting "var". Further details are given in this script.

If you are aiming to display increment of potential temperature at level 18, after modifying "WRFV3/var/graphcs/ncl/WRF-Var_plot.ncl" suitably,

When you execute the following command from "WRFV3/var/graphics/ncl".

```
ncl WRF-Var_plot.ncl
```

The plot created will looks as follows:



Note: Higher the analysis increments, more is the data impact in that region.

What next?

OK, you have run WRF-Var, checked out the diagnostics and are confident things are OK. Now you are ready to run the NWP forecasts using WRF-model.

Updating WRF lateral boundary conditions

Before running NWP forecast using WRF-model, you must modify the tendencies within the lateral boundary condition files to make it consistent with the new WRF-Var initial conditions (analysis). This is absolutely essential because when you initially generated the tendencies for the lateral boundary condition (in wrfbdy_d01 file), it was consistent but subsequently by doing WRF-Var you changed the initial value (at t=0) and so accordingly the initial tendencies needs to be updated in this file (wrfbdy_d01) to adjust the change at the initial time.

This is a simple procedure performed by the WRF utility called "updated_bc".

Note: Make sure that you have "da_update_bc.exe" in "WRFV3/var/da" directory. This executable automatically gets created when you compiled WRF-Var system,

Running UPDATE_BC

The "update_bc.exe" is run via the same wrapper script, you ran con200 by adding following line in your wrapper script.

```
export   RUN_UPDATE_BC=true
```

With this option standard script located at "WRFV3/var/scripts/da_run_update_bc.ksh" will be activated to update the content of "wrfbdy_d01". The new lateral boundary input file will be located at the same location and with the same name. You may like to check its date of creation by issuing the following command

```
ls  –ltr    $RC_DIR/2007010200/wrfbdy_d01
```

What next?

Once you are able to run all these programs successfully, and have spent some time looking at the variety of diagnostics output that is produced, we hope that you'll have some confidence in handling the WRF-Var system programs when you start your cases. Following are some additional WRF-Var exercise to try and learn more about WRF-Var.

Additional WRF-Var Exercises:

(a) Single Observation response in WRF-Var:

In WRF-Var, when you activate the namelist option "pseudo=1" by defining

"export NL_NUM_PSEUDO=1"

WRF-Var generates a single observation with pre-specified *innovation* Observation – First Guess) value at desired location e.g at (in terms of grid co-ordinate) 23x23, level 14 for "U" observation with error characteristics 1 mps, innovation size = 1.0 mps, include following options in your con200 wrapper script and run it.

```
export NL_NUM_PSEUDO=1

export NL_PSEUDO_VAR="u"        #  Can be    "u   u    t    t     q"

export NL_PSEUDO_X=23.0

export NL_PSEUDO_Y=23.0

export NL_PSEUDO_Z=14.0

export NL_PSEUDO_ERR=1.0       #  Should be "1.0 1.0 1.0  1.0  0.001"
```

export NL_PSEUDO_VAL=1.0 # Should be "1.0 1.0 1.0 1.0 0.001"

Also set " export runplot_psot=1"

Note: You may also like to change your "RUN_DIR" to generate output in separate directory. You may also notice that in this sample wrapper script for con200, all these parameters are already set but they are commented. So you need to just remove the undesired comments.

After you have run this new wrapper script successfully, look at various diagnostics files to understand the response of one single "U" observation almost at the middle of the domain. You can display the analysis increments as you did previously or run the same wrapper script by setting:

"export runplot_psot=2"

It will activate some standard NCL script residing in "WRFV3/var/graphics/ncl" and generate following three files in your new $RUN_DIR/plotpsot directory.

xy_psot1.pdf --- XY cross-section

xz_psot1.pdf --- XZ cross-section

yz_psot1.pdf --- YZ cross-section

Note: You may like to repeat this exercise for other observations like temperature ("t"), pressure "ps", specific humidity etc.

(b) Response of BE length scaling parameter:

Run single observation test with following additional parameters

export NL_LEN_SCALING1=0.5 # reduce length psi-scale by 50%

export NL_LEN_SCALING2=0.5 # reduce length chi_u-scale by 50%

export NL_LEN_SCALING3=0.5 # reduce length T-scale by 50%

export NL_LEN_SCALING4=0.5 # reduce length Q-scale by 50%

export NL_LEN_SCALING5=0.5 # reduce length ps-scale by 50%

Note: You may like to try the response of individual variable by setting one parameter at one time. See the spread of analysis increment.

(c) Response of changing BE variance:

Run single observation test with following additional parameters

export NL_VAR_SCALING1=0.25 # reduce psi-variance by 75%

export NL_VAR_SCALING2=0.25 # reduce chi_u-variance by 75%

export NL_VAR_SCALING3=0.25 # reduce T-variance by 75%

export NL_VAR_SCALING4=0.25 # reduce Q-variance by 75%

export NL_VAR_SCALING5=0.25 # reduce ps-variance by 75%

Note: You may like to try the response of individual variable by setting one parameter at one time. See the magnitude of analysis increments.

(d) Response of convergence criteria:

Run con200 case with

export NL_EPS=0.0001

See rsl.out.0000.html and if the solution did not converge in 200 iterations which is the default value for minimization inner iterations loop, increase its limit by setting

export NL_NTMAX=1000.

You may like to compare various diagnostics with earlier run.

(e) Response of outer loop on minimization:

Run con200 case with

export NL_MAX_EXT_ITS=2

With this setting “outer loop” for the minimization procedure will get activated. You may like to compare various diagnostics with earlier run.

Note: Maximum permissible value for “MAX_EXT_ITS” is 10

(f) Response of suppressing a particular type of data in WRF-Var:

Suppose if someone wants not to include “SYNOP” type of data. One way is that do not process it via “obsproc” by not including it in “little_r”. However, it can be suppressed in WRF-Var through namelist option by setting

“export NL_USE_SYNOPOBS=false”

Note: grep “use_” string in “WRFV3/Registry.wrfvar” to learn about all types of data used in WRF-Var.

Chapter 7: WRF Software

Table of Contents

Introduction

WRF Build Mechanism

The WRF build mechanism provides a uniform apparatus for configuring and compiling the WRF model and pre-processors over a range of platforms with a variety of options. This section describes the components and functioning of the build mechanism. For information on building the WRF code, see Section 2.

Required software:

The WRF build relies on Perl version 5 or later and a number of UNIX utilities: csh and Bourne shell, make, M4, sed, awk, and the uname command. A C compiler is needed to compile programs and libraries in the tools and external directories. The WRF code itself is Fortran90. For distributed-memory, MPI and related tools and libraries should be installed.

Build Mechanism Components:

Directory structure: The directory structure of WRF consists of the top-level directory plus directories containing files related to the WRF software framework (frame), the WRF model (dyn_em, phys, share), configuration files (arch, Registry), helper programs (tools), and packages that are distributed with the WRF code (external).

Scripts: The top-level directory contains three user-executable scripts: configure, compile, and clean. The configure script relies on a Perl script in arch/Config.pl.

Programs: A significant number of WRF lines of code are automatically generated at compile time. The program that does this is tools/registry and it is distributed as source code with the WRF model.

Makefiles: The main makefile (input to the UNIX make utility) is in the top-level directory. There are also makefiles in most of the subdirectories that come with WRF. Make is called recursively over the directory structure. Make is not used directly to compile WRF; the compile script is provided for this purpose.

Configuration files: The configure.wrf contains compiler, linker, and other build settings, as well as rules and macro definitions used by the make utility. Configure.wrf is included by the Makefiles in most of the WRF source distribution (Makefiles in tools and external directories do not include configure.wrf). The configure.wrf file in the top-level directory is generated each time the configure script is invoked. It is also deleted by clean -a. Thus, configure.wrf is the place to make temporary changes: optimization levels, compiling with debugging, etc., but permanent changes should be made in arch/configure.defaults.

The arch/configure.defaults file contains lists of compiler options for all the supported platforms and configurations. Changes made to this file will be permanent. This file is used by the configure script to generate a temporary configure.wrf file in the top-level directory. The arch directory also contains the files preamble and postamble, which the unchanging parts of the configure.wrf file that is generated by the configure script.

The Registry directory contains files that control many compile-time aspects of the WRF code (described elsewhere). The files are named Registry.*core*. The configure script copies one of these to Registry/Registry, which is the file that tools/registry will use as input. The choice of *core* depends on settings to the configure script. Changes to Registry/Registry will be lost; permanent changes should be made to Registry.*core*.

Environment variables: Certain aspects of the configuration and build are controlled by environment variables: the non-standard locations of NetCDF libraries or the PERL command, which dynamic core to compile, machine-specific options (e.g. OBJECT_MODE on the IBM systems), etc.

In addition to WRF-related environment settings, there may also be settings specific to particular compilers or libraries. For example, local installations may require setting a variable like MPICH_F90 to make sure the correct instance of the Fortran 90 compiler is used by the mpif90 command.

How the WRF build works:

There are two steps in building WRF: configuration and compilation.

Configuration: The configure script configures the model for compilation on your system. Configure first attempts to locate needed libraries such as NetCDF or HDF and tools such as Perl. It will check for these in normal places, or will use settings from the user's shell environment. Configure then calls the UNIX uname command to discover what platform you are compiling on. It then calls the Perl script in arch/Config.pl, which

traverses the list of known machine configurations and displays a list of available options to the user. The selected set of options is then used to create the configure.wrf file in the top-level directory. This file may be edited but changes are temporary, since the file will be overwritten or deleted by the configure script or clean -a.

Compilation: The compile script is used to compile the WRF code after it has been configured using the configure script, a Csh script that performs a number of checks, constructs an argument list, copies to Registry/Registry the correct Registry.*core* file for the core being compiled, and the invokes the UNIX make command in the top-level directory. The core to be compiled is determined from the user's environment; if no core is specified in the environment (by setting WRF_*CORE*_CORE to 1) the default core is selected (current the Eulerian Mass core). The makefile in the top-level directory directs the rest of the build, accomplished as a set of recursive invocations of make in the subdirectories of WRF. Most of these makefiles include the configure.wrf file in the top-level directory. The order of a complete build is as follows:

1. Make in frame directory
    a. make in external/io_netcdf to build NetCDF implementation of I/O API
    b. make in RSL or RSL_LITE directory to build communications layer (DM_PARALLEL only)
    c. make in external/esmf_time_f90 directory to build ESMF time manager library
    d. make in other external directories as specified by "external:" target in the configure.wrf file
2. Make in the tools directory to build the program that reads the Registry/Registry file and auto-generates files in the inc directory
3. Make in the frame directory to build the WRF framework specific modules
4. Make in the share directory to build the non-core-specific mediation layer routines, including WRF I/O modules that call the I/O API
5. Make in the phys directory to build the WRF model layer routines for physics (non core-specific)
6. Make in the dyn_*core* directory for core-specific mediation-layer and model-layer subroutines
7. Make in the main directory to build the main program(s) for WRF and link to create executable file(s) depending on the build case that was selected as the argument to the compile script (e.g. compile em_real)

8. Symbolic link executable files in the main directory to the run directory for the specific case and to the directory named "run"

Source files (.F and, in some of the external directories, .F90) are preprocessed to produce .f files, which are input to the compiler. As part of the preprocessing, Registry-generated files from the inc directory may be included. Compiling the .f files results in the creation of object (.o) files that are added to the library main/libwrflib.a. The linking step produces the wrf.exe executable and other executables, depending on the case argument to the compile command: real.exe (a preprocessor for real-data cases) or ideal.exe (a preprocessor for idealized cases), and the ndown.exe program, for one-way nesting of real-data cases.

The .o files and .f files from a compile are retained until the next invocation of the clean script. The .f files provide the true reference for tracking down run time errors that refer to line numbers or for sessions using interactive debugging tools such as dbx or gdb.

Registry

Tools for automatic generation of application code from user-specified tables provide significant software productivity benefits in development and maintenance of large applications such as WRF. Some 30-thousand lines of WRF code are automatically generated from a user-edited table, called the Registry. The Registry provides a high-level single-point-of-control over the fundamental structure of the model data, and thus provides considerable utility for developers and maintainers. It contains lists describing state data fields and their attributes: dimensionality, binding to particular solvers, association with WRF I/O streams, communication operations, and run time configuration options (namelist elements and their bindings to model control structures). Adding or modifying a state variable to WRF involves modifying a single line of a single file; this single change is then automatically propagated to scores of locations in the source code the next time the code is compiled.

The WRF Registry has two components: the Registry file, and the Registry program.

The Registry file is located in the Registry directory and contains the entries that direct the auto-generation of WRF code by the Registry program. There may be more than one Registry in this directory, with filenames such as Registry.EM (for builds using the Eulerian Mass core) and Registry.NMM (for builds using the NMM core). The WRF Build Mechanism copies one of these to the file Registry/Registry and this file is used to direct the Registry program. The syntax and semantics for entries in the Registry are described in detail in "WRF Tiger Team Documentation: The Registry" on http://www.mmm.ucar.edu/wrf/WG2/Tigers/Registry/.
The Registry program is distributed as part of WRF in the tools directory. It is built automatically (if necessary) when WRF is compiled. The executable file is tools/registry. This program reads the contents of the Registry file, Registry/Registry, and generates files in the inc directory. These files are included by other WRF source files when they are compiled. Additional information on these is provided as an appendix to "WRF Tiger

Team Documentation: The Registry (DRAFT)". The Registry program itself is written in C. The source files and makefile are in the tools directory.

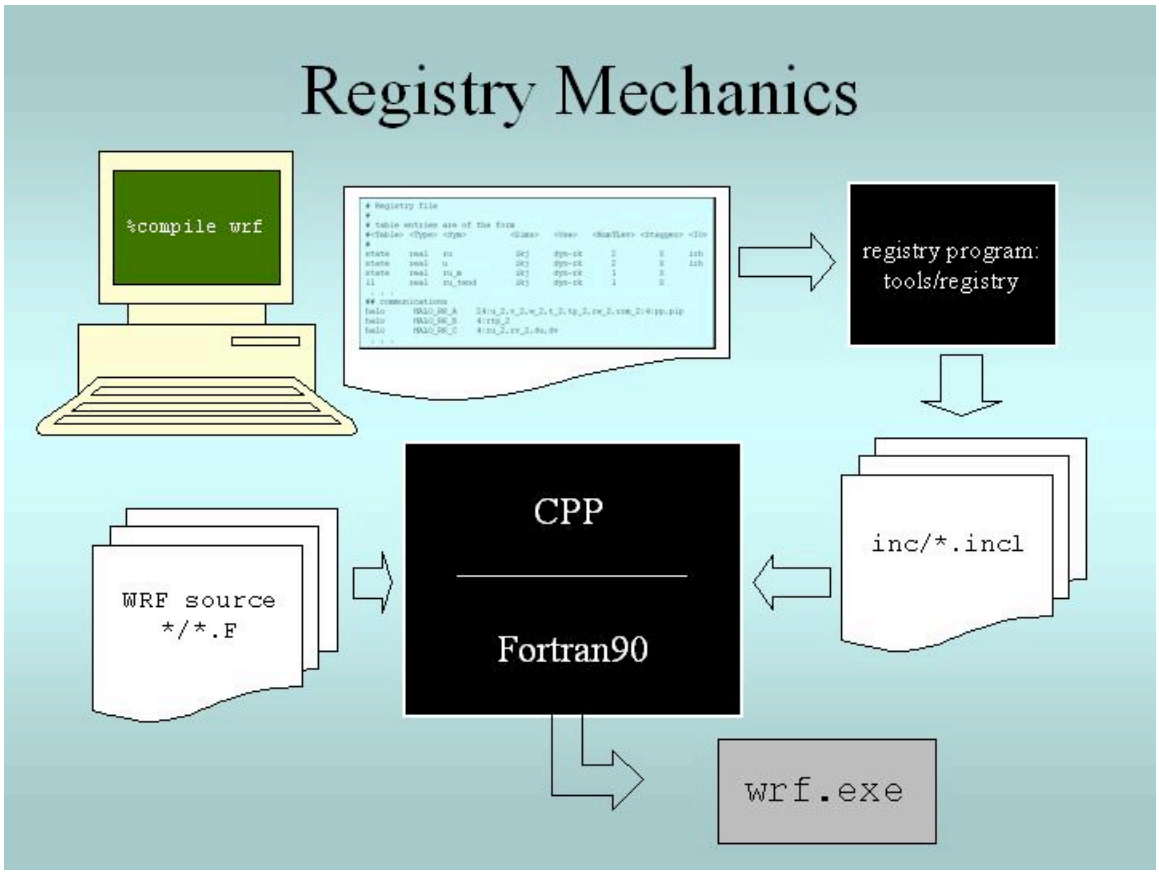


Figure 1. When the user compiles WRF, the Registry Program reads Registry/Registry, producing auto-generated sections of code that are stored in files in the inc directory. These are included into WRF using the CPP preprocessor and the Fortran compiler.

In addition to the WRF model itself, the Registry/Registry file is used to build the accompanying preprocessors such as real.exe (for real data) or ideal.exe (for ideal simulations), and the ndown.exe program (used for one-way, off-line nesting).

I/O Applications Program Interface (I/O API)

The software that implements WRF I/O, like the software that implements the model in general, is organized hierarchically, as a "software stack" (http://www.mmm.ucar.edu/wrf/WG2/Tigers/IOAPI/IOStack.html) . From top (closest to the model code itself) to bottom (closest to the external package implementing the I/O), the I/O stack looks like this:

- Domain I/O (operations on an entire domain)
- Field I/O (operations on individual fields)
- Package-neutral I/O API

- Package-dependent I/O API (external package)

There is additional information on the WRF I/O software architecture on http://www.mmm.ucar.edu/wrf/WG2/IOAPI/IO_files/v3_document.htm. The lower-levels of the stack are described in the I/O and Model Coupling API specification document on http://www.mmm.ucar.edu/wrf/WG2/Tigers/IOAPI/index.html.

Timekeeping

Starting times, stopping times, and time intervals in WRF are stored and manipulated as Earth System Modeling Framework (ESMF, http://www.esmf.ucar.edu) time manager objects. This allows exact representation of time instants and intervals as integer numbers of years, months, hours, days, minutes, seconds, and/or fractions of a second (numerator and denominator are specified separately as integers). All time arithmetic involving these objects is performed exactly, without drift or rounding, even for fractions of a second.

The WRF implementation of the ESMF Time Manger is distributed with WRF in the external/esmf_time_f90 directory. This implementation is entirely Fortran90 (as opposed to the ESMF implementation that required C++) and it is conformant to the version of the ESMF Time Manager API that was available in 2003 (the API has changed in later versions of ESMF and an update will be necessary for WRF once the ESMF specifications and software have stabilized). The WRF implementation of the ESMF Time Manager supports exact fractional arithmetic (numerator and denominator explicitly specified and operated on as integers), a feature needed by models operating at WRF resolutions, but deferred in 2003 since it was not needed for models running at more coarse resolutions.

WRF source modules and subroutines that use the ESMF routines do so by use-association of the top-level ESMF Time Manager module, esmf_mod:

```
USE esmf_mod
```

The code is linked to the library file libesmf_time.a in the external/esmf_time_f90 directory.

ESMF timekeeping is set up on a domain-by-domain basis in the routine setup_timekeeping (share/set_timekeeping.F). Each domain keeps its own clocks, alarms, etc. – since the time arithmetic is exact there is no problem with clocks getting out of synchronization.

Software Documentation

Detailed and comprehensive documentation aimed at WRF software is available at http://www.mmm.ucar.edu/wrf/WG2/software_2.0.

Portability and Performance

WRF is supported on the following platforms:

Vendor	Hardware	OS	Compiler
Apple (*)	G5	MacOS	IBM
Cray Inc.	X1	UNICOS	Cray
HP/Compaq	Alpha	Tru64	Compaq
	Itanium-2	Linux	Intel
		HPUX	HP
IBM	SP Power-3/4	AIX	IBM
SGI	Itanium-2	Linux	Intel
	MIPS	IRIX	SGI
Sun (*)	UltraSPARC	Solaris	Sun
various	Xeon and Athlon	Linux	Intel and Portland Group
	Itanium-2 and Opteron		

Ports are in progress to other systems. Contact wrfhelp@ucar.edu for additional information.

Benchmark information is available at http://www.mmm.ucar.edu/wrf/bench

Chapter 8: Post-Processing Utilities

Table of Contents

Introduction

There are a number of visualization tools available to display WRF-ARW *(http://wrf-model.org/)* model data. Model data in netCDF format, can essentially be displayed using any tool capable of displaying this data format.

Currently the following post-processing utilities are supported, NCL, RIP4, ARWpost *(converter to GrADS and Vis5D)*, WPP, and VAPOR.

NCL, RIP4 and VAPOR can currently only read data in netCDF format, while ARWpost can read data in netCDF and GRIB1 format, and WPP can read data in netCDF and binary format.

Required software

The only library that is always required is the netCDF package from Unidata (***http://www.unidata.ucar.edu/***: login > Downloads > NetCDF - *registration login required*).

netCDF stands for **Net**work **C**ommon **D**ata **F**orm. This format is platform independent, i.e., data files can be read on both big-endian and little-endian computers, regardless of where the file was created. To use the netCDF libraries, ensure that the paths to these libraries are set correct in your login scripts as well as all Makefiles.

Additional libraries required by each of the supported post-processing packages:

- NCL (*http://www.ncl.ucar.edu*)
- GrADS (*http://grads.iges.org/home.html*)
- Vis5D (*http://www.ssec.wisc.edu/~billh/vis5d.html*)
- GEMPAK (*http://my.unidata.ucar.edu/content/software/gempak/index.html*)
- VAPOR (*http://www.vapor.ucar.edu*)

NCL

With the use of **NCL Libraries** *(http://www.ncl.ucar.edu)*, WRF-ARW data can easily be displayed.

The information on these pages has been put together to help users generate NCL scripts to display their WRF-ARW model data.

Some example scripts are available online *(http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/NCL_examples.htm*), but in order to fully utilize the functionality of the NCL Libraries, users should adapt these for their own needs, or write their own scripts.

NCL can process WRF ARW static, input and output files, as well as WRF-Var output data. Both single and double precision data can be processed.

What is NEW?

*In July 2007, the **WRF-NCL** processing scripts have been incorporated into the **NCL Libraries**, thus only the **NCL Libraries**, are now needed. **NCL version 4.3.1** or higher is required. (**NOTE**: Since the release of NCL version 4.3.1, the **WRFUserARW.ncl** script incorporated in the NCL libraries has been updated significantly, and users should get a new version of this script from the **WRF-ARW web** site - http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/Examples/WRFUserARW.ncl).*

With the NCL version 4.3.1 release all WRF related **functions / procedures** needed to plot WRF-ARW are now located in *"$NCARG_ROOT/lib/ncarg/nclscripts/wrf/WRFUsersARW.ncl".*

All the **FORTRAN subroutines** used for diagnostics and interpolation *(previously located in wrf_user_fortran_util_0.f)* has been re-coded into NCL in-line functions. This means users no longer need to compile these routines.

What is NCL

The NCAR Command Language (NCL) is a free interpreted language designed specifically for scientific data processing and visualization. NCL has robust file input and output. It can read in netCDF, HDF4, HDF4-EOS, GRIB, binary and ASCII data. The graphics are world class and highly customizable.

It runs on many different operating systems including Solaris, AIX, IRIX, Linux, MacOSX, Dec Alpha, and Cygwin/X running on Windows. The NCL binaries are freely available at: *http://www.ncl.ucar.edu/Download/*

To read more about NCL, visit: *http://www.ncl.ucar.edu/overview.shtml*

Necessary software

NCL libraries, *version 4.3.1 or higher. Version 5.0.0 is recommended.*

Environment Variable

Set the environment variable NCARG_ROOT to the location where you installed the NCL libraries. Typically *(for cshrc shell):*

```
setenv NCARG_ROOT /usr/local/ncl
```

.hluresfile

Create a file called **.hluresfile** in your $HOME directory. This file controls the color / background / fonts and basic size of your plot. For more information regarding this file, see: *http://www.ncl.ucar.edu/Document/Graphics/hlures.shtml*.

NOTE: *This file must reside in your $HOME directory and not where you plan on running NCL.*

Below is the **.hluresfile** used in the example scripts posted on the web *(scripts are available at: http://www.mmm.ucar.edu/wrf/users/graphics/NCL/NCL.htm*). If a different color table is used, the plots will appear different. Copy the following to your ~/.hluresfile. *(A copy of this file is available at: http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/.hluresfile)*

```
*wkColorMap : BlAqGrYeOrReVi200
*wkBackgroundColor : white
*wkForegroundColor : black
*FuncCode : ~
*TextFuncCode : ~
*Font : helvetica
*wkWidth : 900
*wkHeight : 900
```

NOTE:

If your image has a black background with white lettering, your .hluresfile has not been created correctly, or it is in the wrong location.

*wkColorMap, as set in your .hluresfile can be overwritten in any NCL script with the use of the function "**gsn_define_colormap**", so you do not need to change your .hluresfile if you just want to change the color map for a single plot.*

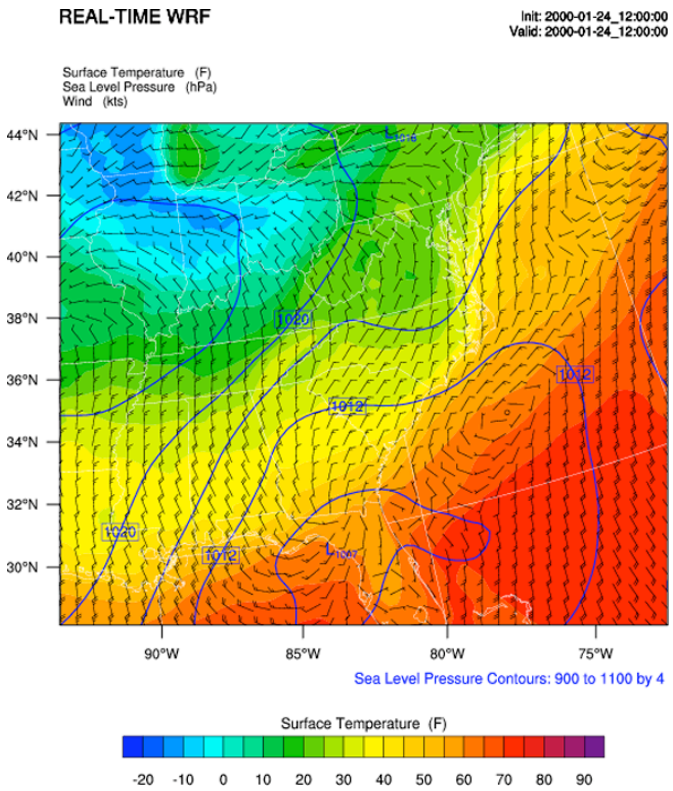
Create NCL scripts

The basic outline of any NCL script will look as follows:

```
load external functions and procedures

begin
      ; Open input file(s)
      ; Open graphical output
      ; Read variables
      ; Set up plot resources & Create plots
      ; Output graphics
end
```

For example, let's create a script to plot Surface Temperature, Sea Level Pressure and Wind as shown in the picture below.



```
; load functions and procedures
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/wrf/WRFUserARW.ncl"

begin

; WRF ARW input file
a = addfile("../wrfout_d01_2000-01-24_12:00:00.nc","r")


; Output on screen. Output will be called "plt_Surface1"
type = "x11"
wks = gsn_open_wks(type,"plt_Surface1")


; Set basic resources
res = True
res@MainTitle = "REAL-TIME WRF"          ; Give plot a main title
res@Footer = False                             ; Set Footers off
pltres = True                                ; Plotting resources
mpres = True                                      ; Map resources

;---------------------------------------------------------------
times = wrf_user_list_times(a)             ; get times in the file
it = 0                                ; only interested in first time
res@TimeLabel = times(it)                ; keep some time information

;---------------------------------------------------------------
; Get variables

slp = wrf_user_getvar(a,"slp",it)                            Get slp
   wrf_smooth_2d( slp, 3 )                                ; Smooth slp

t2 = wrf_user_getvar(a,"T2",it)                    ; Get T2 (deg K)
   tc2 = t2-273.16                                 ; Convert to deg C
   tf2 = 1.8*tc2+32.                               ; Convert to deg F
   tf2@description = "Surface Temperature"
   tf2@units = "F"

u10 = wrf_user_getvar(a,"U10",it)                           ; Get U10
v10 = wrf_user_getvar(a,"V10",it)                           ; Get V10
   u10 = u10*1.94386                               ; Convert to knots
   v10 = v10*1.94386
   u10@units = "kts"
   v10@units = "kts"

;---------------------------------------------------------------
```

```
; Plotting options for T
opts = res                                        ; Add basic resources
opts@cnFillOn = True                                    ; Shaded plot
opts@ContourParameters = (/ -20., 90., 5./)    ; Contour intervals
opts@gsnSpreadColorEnd = -3
contour_tc = wrf_contour(a,wks,tf2,opts)              ; Create plot
delete(opts)


; Plotting options for SLP
opts = res                                        ; Add basic resources
opts@cnLineColor = "Blue"                           ; Set line color
opts@cnHighLabelsOn = True                             ; Set labels
opts@cnLowLabelsOn = True
opts@ContourParameters = (/ 900.,1100.,4./)    ; Contour intervals
contour_psl = wrf_contour(a,wks,slp,opts)             ; Create plot
delete(opts)


; Plotting options for Wind Vectors
opts = res                                        ; Add basic resources
opts@FieldTitle = "Winds"                  ; Overwrite the field title
opts@NumVectors = 47                          ; Density of wind barbs
vector = wrf_vector(a,wks,u10,v10,opts)               ; Create plot
delete(opts)


; MAKE PLOTS
plot = wrf_map_overlays(a,wks,  \
      (/contour_tc,contour_psl,vector/),pltres,mpres)

;----------------------------------------------------------------


end
```

Extra sample scripts are available at,
http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/NCL_examples.htm

Run NCL scripts

1. Ensure NCL is successfully installed on your computer.

2. Ensure that the environment variable NCARG_ROOT is set to the location where NCL is installed on your computer. Typically *(for cshrc shell),* the command will look as follows:

```
setenv NCARG_ROOT /usr/local/ncl
```

3. Create an NCL plotting script.

4. Run the NCL script you created:

```
ncl  NCL_script
```

The output type created with this command is controlled by the line:
wks = gsn_open_wk (type,"Output") ; inside the NCL script
where *type* can be *x11, pdf, ncgm, ps,* or *eps*

For high quality images, create pdf / ps or eps images directly via the ncl scripts **(*type* = *pdf / ps / eps*)**

See the **Tools** section at the end of this chapter for more information concerning other types of graphical formats and conversions between graphical formats.

Functions / Procedures under "$NCARG_ROOT/lib/ncarg/nclscripts/wrf/" *(WRFUserARW.ncl)*

function wrf_user_getvar (*nc_file, fld, it*)
Usage: *ter = wrf_user_getvar (a, "HGT", 0)*

Get fields from netCDF file for any given time.
Currently only a single time can be extracted with each call to ***wrf_user_getvar.***

Any field available in the netCDF file can be extracted.
fld is case sensitive. The policy adapted during development was to set all diagnostic variables calculated by NCL to lower-case to distinguish them from fields directly available from the netCDF files.

List of available diagnostics *(currently most of these only work of wrfout files)*:
th – Potential Temperature *(K)*
tk – Temperature *(K)*
tc – Temperature *(C)*
td – Dewpoint Temperature *(C)*
td2 – 2m Dewpoint Temperature *(C)*
rh – Relative Humidity *(%)*
slp – Sea Level Pressure *(hPa)*
pressure – Full model pressure *(hPa)*
z - Geopotential Height *(m)*
ua - U-component of the wind on mass points *(un-staggered)*
va - V-component of the wind on mass points *(un-staggered)*
wa - W-component of the wind on mass points *(un-staggered)*

uvmet – winds rotated to earth coordinates. Used specifically for sounding and to compare model winds to observations. This diagnostics is unique as it returns a 4D field, where uvmet(0,:,:,:) contains the U-component and uvmet(1,:,:,:) contains the V-component of the wind *(un-staggered)*.

function wrf_user_list_times (*nc_file*)
Usage: *times* = wrf_user_list_times *(a)*

Obtain a list of times available in the input file. The function returns a 1D array containing the times *(type: character)* in the input file.

function wrf_contour (*nc_file, wks, data, res*)
Usage: *contour* = wrf_contour *(a, wks, ter, opts)*

Returns a graphic (*contour*), of the ***data*** to be contoured. This graphic is only created, but not plotted to a wks. This enables a user to generate many such graphics and overlay them before plotting the resulting picture to a wks.

The returned graphic (*contour*) does not contain map information, and can therefore be used for both real and idealized data cases.

This function can plot both line contours and shaded contours. *Default is line contours.*

Many resources are set for a user, of which most can be overwritten. Below is a list of resources you may want to consider changing before generating your own graphics:

opts@MainTitle : Controls main title on the plot.
opts@MainTitlePos : Main title position – Left/Right/Center. *Default is Left.*
opts@NoHeaderFooter : Switch off all Headers and Footers.
opts@Footer : Add some model information to the plot as a footer. *Default is True*.
opts@InitTime : Plot initial time on graphic. *Default is True*. If True, the initial time will be extracted from the input file.
opts@ValidTime : Plot valid time on graphic. *Default is True*. A user must set opts@TimeLabel to the correct time.
opts@TimeLabel : Time to plot as valid time.
opts@TimePos : Time position – Left/Right. *Default is "Right".*
opts@cnFillOn : Set to True for shaded plots. Default is False.
opts@ContourParameters : A single value is treated as an interval. Three values represent: Start, End, and Interval.
opts@cnLineColor : Color of line plot.
opts@FieldTitle : Overwrite the field title.
opts@lbTitleOn : Set to False to switch the title on the label bar off. *Default is True*.

optr@cnLevelSelectionMode ; opts @cnLevels ; opts@cnFillColors ;
optr@cnConstFLabelOn : Can be used to set contour levels and colors manually.

function wrf_vector (*nc_file, wks, data_u, data_v, res*)
Usage: *vector* = wrf_vector *(a, wks, ua, va, opts)*

Returns a graphic (*vector*) of the data. This graphic is only created, but not plotted to a wks. This enables a user to generate many graphics and overlay them before plotting the resulting picture to a wks.

The returned graphic (*vector*) does not contain map information, and can therefore be used for both real and idealized data cases.

Many resources are set for a user, of which most can be overwritten. Below is a list of resources you may want to consider changing before generating your own graphics:

opts@MainTitle ; opts@MainTitlePos ; opts@NoHeaderFooter ; opts@Footer ;
opts@InitTime ; opts@ValidTime ; opts@TimeLabel ; opts@TimePos ;
opts@FieldTitle : Applies and are the same as described for *wrf_contour.*
opts@NumVectors : Density of wind vectors.
opts@vcGlyphStyle : Wind style. *"WindBarb" is default*.

function wrf_map_overlays (*nc_file, wks, (/graphics/), pltres, mpres*)
Usage: *plot = wrf_map_overlays (a, wks, (/contour,vector/), pltres, mpres)*

Overlay contour and vector plots generated with *wrf_contour* and *wrf_vector*. Can overlay any number of graphics. Overlays will be done in order give, so always list shaded plots before line or vector plots, to ensure the lines and vectors are visible and not hidden behind the shaded plot.

A map background will automatically be added to the plot. Map details are controlled with the *mpres* resource. Common map resources you may want to set are:
mpres@mpGeophysicalLineColor ; mpres@mpNationalLineColor ;
mpres@mpUSStateLineColor ; mpres@mpGridLineColor ;
mpres@mpLimbLineColor ; mpres@mpPerimLineColor

If you want to zoom into the plot, set *mpres@ZoomIn* to True, and *mpres@Xstart, mpres@Xend, mpres@Ystart, mpres@Yend*, to the corner x/y positions of the zoomed plot.

pltres@NoTitles : Set to True to remove all field titles on a plot.

pltres@CommonTitle : Overwrite field titles with a common title for the overlaid plots.
Must set *pltres@PlotTitle* to desired new plot title.

If you want to generate images for a panel plot, set *pltres@PanelPot* to True.

If you want to add text/lines to the plot before advancing the frame, set *pltres@FramePlot* to False. Add your text/lines directly after the call to the *wrf_map_overlays* function. Once you are done adding text/lines, advance the frame with the command "***frame(wks)***".

function wrf_overlays (nc_file, *wks, (/graphics/), pltres*)
Usage: *plot = wrf_overlays (a, wks, (/contour,vector/), pltres)*

Overlay contour and vector plots generated with *wrf_contour* and *wrf_vector*. Can overlay any number of graphics. Overlays will be done in order give, so always list shaded plots before line or vector plots, to ensure the lines and vectors are visible and not hidden behind the shaded plot.

Typically used for idealized data or cross-sections, which does not have map background information.

pltres@NoTitles : Set to True to remove all field titles on a plot.
pltres@CommonTitle : Overwrite field titles with a common title for the overlaid plots.
Must set *pltres@PlotTitle* to desired new plot title.

If you want to generate images for a panel plot, set *pltres@PanelPot* to True.

If you want to add text/lines to the plot before advancing the frame, set *pltres@FramePlot* to False. Add your text/lines directly after the call to the *wrf_overlays* function. Once you are done adding text/lines, advance the frame with the command "***frame(wks)***".

function wrf_map (*nc_file, wks, res*)
Usage: *map = wrf_map (a, wks, opts)*

Create a map background.
As maps are added to plots automatically via the *wrf_map_overlays* function, this function is seldom needed as a stand-alone.

function wrf_user_intrp3d (*var3d, H, plot_type, loc_param, angle, res*)

This function is used for both horizontal and vertical interpolation.

var3d: The variable to interpolate.
H: The field to interpolate to. Either *pressure* or *z*.
plot_type: "h" for horizontally and "v" for vertically interpolated plots.
loc_param: Can be a scalar, or an array holding either 2 or 4 values.
For *plot_type* = "h", this is a scalar representing the level to interpolate too, i.e., 500 to interpolate to 500 hPa; or 2000 to interpolate to 2 km.
For *plot_type* = "v": This can be a pivot point though which a line is drawn – in this case a single x/y point (*2 values*) is required. Or this can be a set of x/y points (*4 values*), indicating start x/y and end x/y locations for the cross-section.
angle: Set to 0., for *plot_type* = "h", or for *plot_type* = "v" when start and end locations of cross-section were supplied in *loc_param*.
If a single pivot point was supplied in *loc_param*, *angle* is the angle of the line that will pass through the pivot point. Where: 0. is SN, and 90. is WE.
res: Set to False for *plot_type* = "h", or for *plot_type* = "v" when a single pivot point is supplied. Set to True if start and end locations is supplied.

function wrf_user_intrp2d (*var2d, loc_param, angle, res*)

This function interpolates a 2D field along a given line.

var2d: Is the 2D field to interpolate.

loc_param: Is an array holding either 2 or 4 values.
This can be a pivot point though which a line is drawn – in this case a single x/y point (*2 values*) is required. Or this can be a set of x/y points (*4 values*), indicating start x/y and end x/y locations for the cross-section.
angle: Set to 0 when start and end locations of the line were supplied in *loc_param*.
If a single pivot point was supplied in *loc_param*, *angle* is the angle of the line that will pass through the pivot point. Where: 0. is SN, and 90. is WE.
res: Set to False when a single pivot point is supplied. Set to True if start and end locations is supplied.

function wrf_user_latlon_to_ij (*nc_file, lats, lons*)
Usage: *loc = wrf_user_latlon_to_ij (a, 40., 100.)*
Usage: *loc = wrf_user_latlon_to_ij (a, (/40., 50./), (/100., 120./))*

Convert a lon/lat location to the nearest x/y location. Note: no interpolation is done, only the closest grid point will be returned.

lats/lons can be scalars or arrays.
loc(:,0) is the y (SN) location, and loc(:,1) the x (WE) location.

Adding diagnostics using FORTRAN code

It is possible to link your favorite FORTRAN diagnostics routines to NCL. It is easier to use FORTRAN 77 code, but NCL does recognize basic FORTRAN 90 code.

Let's use a routine that calculated temperature (K) from theta and pressure.

FORTRAN 90 routine called myTK.f90

```
subroutine compute_tk (tk, pressure, theta, nx, ny, nz)
implicit none

!! Variables
   integer  :: nx, ny, nz
   real, dimension (nx,ny,nz) :: tk, pressure, theta

!! Local Variables
   integer :: i, j, k
   real, dimension (nx,ny,nz):: pi

   pi(:,:,:) = (pressure(:,:,:) / 1000.)**(287./1004.)
   tk(:,:,:) = pi(:,:,:)*theta(:,:,:)

return
end subroutine compute_tk
```

For simple routines like this, it is easiest to re-write the routine into a FORTRAN 77 routine.

FORTRAN 77 routine called myTK.f

```
      subroutine compute_tk (tk, pressure, theta, nx, ny, nz)
      implicit none

C     Variables
      integer  nx, ny, nz
      real   tk(nx,ny,nz) , pressure(nx,ny,nz), theta(nx,ny,nz)

C     Local Variables
      integer  i, j, k
      real   pi
```

```
      DO k=1,nz
        DO j=1,ny
          DO i=1,nx
            pi=(pressure(i,j,k) / 1000.)**(287./1004.)
            tk(i,j,k) = pi*theta(i,j,k)
          ENDDO
        ENDDO
      ENDDO

    return
    end
```

Add the markers **NCLFORTSTART** and **NCLEND** to the subroutine as indicated below. Note, that local variables are outside these block markers.

FORTRAN 77 routine called myTK.f, with NCL markers added

```
C NCLFORTSTART
      subroutine compute_tk (tk, pressure, theta, nx, ny, nz)
      implicit none

C     Variables
      integer  nx, ny, nz
      real   tk(nx,ny,nz) , pressure(nx,ny,nz), theta(nx,ny,nz)

C NCLEND

C     Local Variables
      integer  i, j, k
      real   pi

      DO k=1,nz
        DO j=1,ny
          DO i=1,nx
            pi=(pressure(i,j,k) / 1000.)**(287./1004.)
            tk(i,j,k) = pi*theta(i,j,k)
          ENDDO
        ENDDO
      ENDDO

    return
    end
```

Now compile this code using the NCL script WRAPIT.

```
WRAPIT myTK.f
```

NOTE: If WRAPIT cannot be found, make sure the environment variable *NCARG_ROOT* has been set correctly.

If the subroutine compiles successfully, a new library will be created, called **myTK.so.** This library can be linked to an NCL script to calculate TK. See how this is done in the example below:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/wrf/WRFUserARW.ncl"
external myTK "./myTK.so"

begin

        t = wrf_user_getvar (a,"T",5)
        theta = t + 300
        p = wrf_user_getvar (a,"pressure",5)

        dim = dimsizes(t)
        tk = new( (/ dim(0), dim(1), dim(2) /), float)

        myTK :: compute_tk (tk, p, theta, dim(2), dim(1), dim(0))

end
```

Want to use the FORTRAN 90 program? It is possible to do so by providing an interface block for your FORTRAN 90 program. Your FORTRAN 90 program may also not contain any of the following features:

- pointers or structures as arguments,
- missing/optional arguments,
- keyword arguments, or
- if the procedure is recursive.

Interface block for FORTRAN 90 code, called myTK90.stub

```
C NCLFORTSTART
      subroutine compute_tk (tk, pressure, theta, nx, ny, nz)

      integer  nx, ny, nz
      real   tk(nx,ny,nz) , pressure(nx,ny,nz), theta(nx,ny,nz)

C NCLEND
```

Now compile this code using the NCL script WRAPIT.

```
WRAPIT myTK90.stub myTK.f90
```

NOTE: You may need to copy the WRAPIT script to a locate location and edit it to point to a FORTRAN 90 compiler.

If the subroutine compiles successfully, a new library will be created, called **myTK90.so** *(note the change in name from the FORTRAN 77 library).* This library can similarly be linked to an NCL script to calculate TK. See how this is done in the example below:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/wrf/WRFUserARW.ncl”
external myTK90 "./myTK90.so"

begin

        t = wrf_user_getvar (a,”T”,5)
        theta = t + 300
        p = wrf_user_getvar (a,”pressure”,5)

        dim = dimsizes(t)
        tk = new( (/ dim(0), dim(1), dim(2) /), float)

        myTK90 :: compute_tk (tk, p, theta, dim(2), dim(1), dim(0))

end
```

RIP4

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 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.*

The RIP **users' guide** (*http://www.mmm.ucar.edu/wrf/users/docs/ripug.htm*) is essential reading.

Code history

Version 4.0: reads WRF-ARW real output files
Version 4.1: reads idealized WRF-ARW datasets
Version 4.2: reads all the files produced by WPS
Version 4.3: reads files produced by WRF-NMM model
(This document will only concentrate on running RIP4 for WRF-ARW. For details on running RIP4 for WRF-NMM, see the WRF-NMM User's Guide: http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/WPS/index.php)

Necessary software

RIP4 only requires low level NCAR Graphics libraries. These libraries have been merged with the NCL libraries since the release of NCL version 5 (*http://www.ncl.ucar.edu/*), so if you don't already have NCAR Graphics installed on your computer, install NCL version 5.

Obtain the code from the WRF-ARW user's web site:
http://www.mmm.ucar.edu/wrf/users/download/get_source.html

Unzip and untar the RIP4 tar file. 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 Users' Guide (*ripug*).
- *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.
- *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.
- *src/*, a directory that contains all of the source code files for RIP, RIPDP, and several other utility programs.
- *stationlist*, a file containing observing station location information.

Environment Variables

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.
Typically *(for cshrc shell):*

```
setenv RIP_ROOT /my-path/RIP4
```

The RIP_ROOT environment variable can also be overwritten with the variable *rip_root* in the RIP user input file (UIF).

A second environment variable you need to set is **NCARG_ROOT**.
Typically *(for cshrc shell):*

```
setenv NCARG_ROOT /usr/local/ncarg     ! for NCARG V4
setenv NCARG_ROOT /usr/local/ncl       ! for NCL V5
```

Compiling RIP and associated programs

Typing "make" will produce a list of available compile options *(the list shown below is just a sample of what is available)*:

```
make dec        To Run on DEC_ALPHA
make linux      To Run on LINUX with PGI compiler
make intel      To Run on LINUX with INTEL compiler
make sun        To Run on SUN
make sgi        To Run on SGI
make ibm        To Run on IBM SP2
make cray       To Run on NCAR's Cray
make clean      to remove object files
make clobber    to remove object files and executables
```

Pick the compiler option for the machine you are working on and type:
"make *machine*"

e.g. *make linux*

will compile the code for a Linux computer running PGI compiler.

After a successful compilation, the following new files should be created.

rip	RIP post-processing program. Before using this program, first convert the input data to the correct format expected by this program, using the program ripdp
ripcomp	This program reads in two rip data files and compares their contents.
ripdp_mm5	RIP Data Preparation program for MM5 data
ripdp_wrfarw **ripdp_wrfnmm**	RIP Data Preparation program for WRF data
ripinterp	This program reads in model output (in rip-format files) from a coarse domain and from a fine domain, and creates a new file which has the data from the coarse domain file interpolated (bi-linearly) to the fine domain. The header and data dimensions of the new file will be that of the fine domain, and the case name used in the file name will be the same as that of the fine domain file that was read in.
ripshow	This program reads in a rip data file and prints out the contents of the header record.
showtraj	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. showtraj, reads the trajectory position file and prints out its contents in a readable form. When you run showtraj, it prompts you for the name of the trajectory position file to be printed out.
tabdiag	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. tabdiag serves this purpose.
upscale	This program reads in model output (in rip-format files) from a coarse domain and from a fine domain, and replaces the coarse data with fine data at overlapping points. Any refinement ratio is allowed, and the fine domain borders do not have to coincide with coarse domain grid points.

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 RIP Data Preparation). RIPDP reads in a model output file (or files), and separates out each variable at each time.

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 "mm5", "wrfarw", or "wrfnmm".
The argument model-data-set-name can be any string you choose, that uniquely defines this model output data set

The use of the namelist file is optional. The most important information in the namelist, is the times you want to process.

As this step will create a large number of extra files, creating a new directory to place these files in, will enable you to manage the files easier *(`mkdir RIPDP`)*.

e.g. `ripdp_wrfarw RIPDP/arw all wrfout_d01_*`

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 *(see Chapter 4 of the RIP users' guide for details)*. 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 (***userin*** - *which control the general input specifications; and* ***trajcalc*** - *which control the creation of trajectories*). The second section is the plot specification section, which is used to specify which plots will be generated.

namelist: userin

Variable	Value	Description
idotitle	1	Control first part of title.
title	'auto'	Define your own title, or allow RIP to generate one.
titlecolor	'def.foreground'	Control color of the title.
iinittime	1	Print initial date and time *(in UTC)* on plot.
ifcsttime	1	Print forecast lead-time *(in hours)* on plot.
ivalidtime	1	Print valid date and time (*in both UTC and local time*) on plot.
inearesth	0	This 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	-7.0	Specifies the offset from Greenwich time.
iusdaylightrule	1	Flag to determine if US daylight saving should be applied.
ptimes	9.0E+09	Times to process. This can be a string of times *(e.g. 0,3,6,9,12,)* or a series in the form of *A,-B,C,* which means "times from hour *A*, to hour *B*, every *C* hours" *(e.g. 0,-12,3,)*. Either *ptimes* or *iptimes* can be used, but not both. *You can plot all available times, by omitting both ptimes and iptimes from the namelist, or by setting the first value negative.*
ptimeunits	'h'	Time units. This can be 'h' *(hours)*, 'm' *(minutes)*, or 's' *(seconds). Only valid with ptimes.*
iptimes	99999999	Times to process. This is an integer array that specifies desired times for RIP to plot, but in the form of 8-digit "mdate" times *(i.e. YYMMDDHH)*. Either *ptimes* or *iptimes* can be used, but not both. *You can plot all available times, by omitting both ptimes and iptimes from the namelist, or by setting the first value negative.*
tacc	1.0	Time tolerance in seconds. Any time in the model output that is within *tacc* seconds of the time specified in *ptimes/iptimes* will be processed.
flmin, flmax, fbmin, ftmax	.05, .95, .10, .90	Left, right, bottom and top frame limit
ntextq	0	Text quality specifier (0=high; 1=medium; 2=low).

ntextcd	0	Text font specifier [0=complex (Times); 1=duplex (Helvetica)].
fcoffset	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. Examples: *fcoffset*=12 means you consider hour 12 in the model output to be the beginning of the true forecast.
idotser	0	Generate time series output files *(no plots)* only an ASCII file that can be used as input to a plotting program.
idescriptive	1	Use more descriptive plot titles.
icgmsplit	0	Split metacode into several files.
maxfld	10	Reserve memory for RIP.
ittrajcalc	0	Generate trajectory output files (use namelist *trajcalc* when this is set).
imakev5d	0	Generate output for Vis5D
ncarg_type	'cgm'	Output type required. Options are 'cgm' *(default)*, 'ps', 'pdf', 'pdfL', 'x11'. Where 'pdf' is portrait and 'pdfL' is landscape.
istopmiss	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	'/dev/null'	Overwrite the environment variable RIP_ROOT.

Plot Specification Table

The second part of the RIP UIF consists of the Plot Specification Table. The PST provides all of the user control over particular aspects of individual frames and overlays.

The basic structure of the PST is as follows:

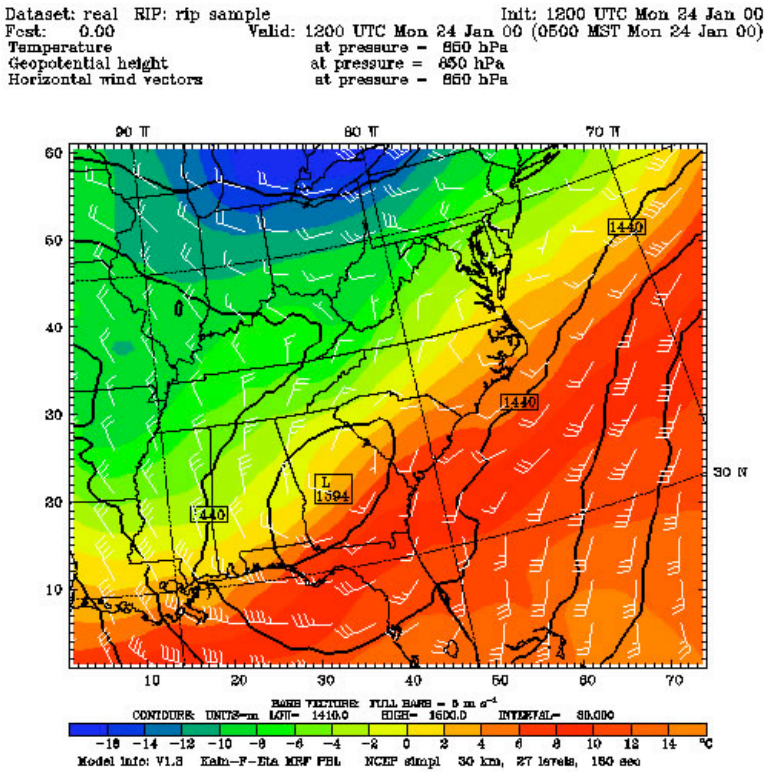
- The first line of the PST is a line of consecutive equal signs. This line as well as the next two lines is ignored by RIP, it is simply a banner that says this is the start of the PST section.
- After that there are several groups of one or more lines separated by a full line of equal signs. Each group of lines is a frame specification group (FSG), and it describes what will be plotted in a single frame of metacode. Each FSG must end with a full line of equal signs, so that RIP can determine where individual frames starts and ends.
- Each line within a FGS is referred to as a plot specification line (PSL). A FSG that consists of three PSL lines will result in a single metacode frame with three overlaid plots.

Example of a frame specification groups (FSG's):

```
=================================================
 feld=tmc; ptyp=hc; vcor=p; levs=850; >
   cint=2; cmth=fill; cosq=-32,light.violet,-24,
   violet,-16,blue,-8,green,0,yellow,8,red,>
   16,orange,24,brown,32,light.gray
feld=ght; ptyp=hc; cint=30; linw=2
feld=uuu,vvv; ptyp=hv; vcmx=-1; colr=white; intv=5
feld=map; ptyp=hb
feld=tic; ptyp=hb
=================================================
```

This **FSG** will generate 5 frames to create a single plot (as shown below):

- Temperature in degrees C (*feld=**tmc***). This will be plotted as a horizontal contour plot (*ptyp=hc*), on pressure levels (*vcor=p*). The data will be interpolated to 850 hPa. The contour intervals are set to 2 (*cint=2*), and shaded plots (*cmth=fill*) will be generated with a color range from light violet to light gray.
- Geopotential heights (*feld=**ght***) will also be plotted as a horizontal contour plot. This time the contour intervals will be 30 (*cint=30*), and contour lines, with a line width of 2 (*linw=2*) will be used.
- Wind vectors (*feld=**uuu,vvv***), plotted as barbs (*vcmax=-1*).
- A map background will be displayed (*feld=**map***), and
- Tic marks will be placed on the plot (*feld=tic*).



Running RIP

Each execution of RIP requires three basic things: a RIP executable, a model data set and a user input file (UIF). 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*.

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 *–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.

```
e.g.  rip  -f  RIPDP/arw  rip_sample
```

If this is successful, the following files will be created:

rip_sample.*TYPE* - metacode file with requested plots
rip_sample.out - log file *(if –f used)* ; view this file if a problem occurred

The default output *TYPE* is 'cgm', metacode file. To view these, use the command 'idt'.

```
e.g.  idt   rip_sample.cgm
```

For high quality images, create pdf or ps images directly **(ncarg_*type = pdf / ps*).**

See the **Tools** section at the end of this chapter for more information concerning other types of graphical formats and conversions between graphical formats.

Examples of plots created for both idealized and real cases are available from:
http://www.mmm.ucar.edu/wrf/users/graphics/RIP4/RIP4.htm

ARWpost

The ARWpost package reads in WRF-ARW model data and creates output in either GrADS or Vis5D format.

The converter can read in WPS geogrid and metgrid data, and WRF-ARW input and output files.

The package makes use of the WRF IO API. The netCDF format has been tested extensively. GRIB1 format has been tested, but not as extensively. BINARY data cannot be read at the moment.

Necessary software

GrADS software - you can download and install GrADS from *http://grads/iges.org/grads*. The GrADS software is not needed to compile and run ARWpost.

Vis5D software (*http://www.ssec.wisc.edu/~billh/vis5d.html*)
Vis5D libraries must be installed to compile and run the ARWpost code, when creating Vis5D input data. If Vis5D files are not being created, these libraries are NOT needed to compile and run ARWpost.

Obtain the ARWpost TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
WRFV3 must be installed and available somewhere, as ARWpost makes use of the common IO API libraries from WRFV3.

Unzip and untar the ARWpost tar file.

The tar file contains the following directories and files:

- *README*, a text file containing basic information on running ARWpost.
- *arch/,* directory containing configure and compilation control.
- *clean,* a script to clean compiled code.
- *compile*, a script to compile the code.
- *configure*, a script to configure the compilation for your system.
- *namelist.ARWpost,* namelist to control the running of the code.
- *src/*, directory containing all source code.
- *scripts/*, directory containing some grads sample scripts.
- gribinfo.txt & gribmap.txt, files needed to process GRIB1 data. Do not edit these files.
- *util/*, a directory containing some utilities.

Environment Variables

Set the environment variable NETCDF to the location where your netCDF libraries are installed. Typically *(for cshrc shell):*

```
setenv NETCDF /usr/local/netcdf
```

Configure ARWpost

WRFV3 must be compiled and available on your system.

Type:

```
./configure
```

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

```
Will use NETCDF in dir: /usr/local/netcdf-pgi
------------------------------------------------------------
Please select from among the following supported platforms.
1. PC Linux i486 i586 i686, PGI compiler (no vis5d)
2. PC Linux i486 i586 i686, PGI compiler (vis5d)
3. PC Linux i486 i586 i686, Intel compiler (no vis5d)
4. PC Linux i486 i586 i686, Intel compiler (vis5d)

Enter selection [1-4]
```

Make sure the netCDF path is correct.
Pick compile options for your machine *(if you do not have Vis5D, or if you do not plan on using it, pick an option without Vis5D libraries)*.

Compile ARWpost

If your **WRFV3** code is NOT compiled under **../WRFV3**, edit configure.arwp, and set **"WRF_DIR"** to the correct location of your WRFV3 code.
Type**:**

```
./compile
```

If successful, the executable `ARWpost.exe` will be created.

Edit the namelist.ARWpost file

Set input and output file names and fields to process (***&io***)

Variable	Value	Description
&datetime		
start_date; *end_date*	-	Start and end dates to process. Format: YYYY-MM-DD_HH:00:00
interval_seconds	0	Interval in seconds between data to process. If data is available every hour, and this is set to every 3 hours, the code will skip past data not required.
tacc	0	Time tolerance in seconds. Any time in the model output that is within *tacc* seconds of the time specified will be processed.
debug_level	0	Set higher to debugging is required.
&io		
io_form_input	-	2=netCDF, 5=GRIB1
input_root_name	./	Path and root name of files to use as input. All files starting with the root name will be processed. Wild characters are allowed.
output_root_name	./	Output root name. When converting data to GrADS, *output_root_name*.ctl and *output_root_name*.dat will be created. For Vis5D, *output_root_name*.v5d will be created.
output_title	Title as in WRF file	Use to overwrite title used in GrADS .ctl file.
mercator_defs	.False.	Set to true if mercator plots are distorted.
output_type	'grads'	Options are 'grads' or 'v5d'
split_output	.False.	Use if you want to split our GrADS output files into a number of smaller files *(a common .ctl file will be used for all .dat files).*
frames_per_outfile	1	If *split_output* is .True., how many time periods are required per output *(.dat)* file.
plot	'all'	Which fields to process. 'all' – all fields in WRF file 'list' – only fields as listed in the *'fields'* variable. 'all_list' – all fields in WRF file and all fields listed in the *'fields'* variable.

		Order has no effect, i.e., *'all_list'* and *'list_all'* are similar. If '*list*' is used, a list of variables must be supplied under '*fields*'. Use *'list'* to calculate diagnostics.
fields	-	Fields to plot. Only used if '*list'* was used in the '*plot*' variable.
&interp		
interp_method	0	0 - sigma levels, -1 - code defined "nice" height levels, 1 - user defined height or pressure levels
interp_levels	-	Only used if interp_method=1 Supply levels to interpolate to, in hPa (pressure) or km (height). Supply levels bottom to top.

Available diagnostics:

cape - 3d cape
cin - 3d cin
mcape - maximum cape
mcin - maximum cin
clfr - low/middle and high cloud fraction
dbz - 3d reflectivity
max_dbz - maximum reflectivity
height - model height in km
lcl - lifting condensation level
lfc - level of free convection
pressure - full model pressure in hPa
rh - relative humidity
rh2 - 2m relative humidity
theta - potential temperature
tc - temperature in degrees C
tk - temperature in degrees K
td - dew point temperature in degrees C
td2 - 2m dew point temperature in degrees C
slp - sea level pressure
umet and **vmet** - winds rotated to earth coordinates
u10m and **v10m** - 10m winds rotated to earth coordinates
wdir - wind direction
wspd - wind speed coordinates
wd10 - 10m wind direction
ws10 - 10m wind speed

Run ARWpost

Type:

```
./ARWpost.exe
```

This will create *output_root_name*.**dat** and *output_root_name*.**ctl** files if creating GrADS input, and *output_root_name*.**v5d**, if creating Vis5D input.

NOW YOU ARE READY TO VIEW THE OUTPUT

GrADS

For general information about working with GrADS, view the GrADS home page: *http://grads.iges.org/grads/*

To help users get started a number of GrADS scripts have been provided:

- The scripts are all available in the scripts/ directory.
- The scripts provided are only examples of the type of plots one can generate with GrADS data.
- The user will need to modify these scripts to suit their data (e.g., if you did not specify 0.25 km and 2 km as levels to interpolate to when you run the "bwave" data through the converter, the "bwave.gs" script will not display any plots, since it will specifically look for these to levels).
- Scripts must be copied to the location of the input data.

GENERAL SCRIPTS

cbar.gs	Plot color bar on shaded plots (from GrADS home page)
rgbset.gs	Some extra colors *(Users can add/change colors from color number 20 to 99)*
skew.gs	Program to plot a skewT
	TO RUN TYPE: run skew.gs (needs pressure level TC,TD,U,V as input) User will be prompted if a hardcopy of the plot must be create - 1 for yes and 0 for no. If 1 is entered, a GIF image will be created. Need to enter lon/lat of point you are interested in Need to enter time you are interested in Can overlay 2 different times

plot_all.gs Once you have opened a GrADS window, all one needs to do is run this script.
It will automatically find all .ctl files in the current directory and list them so one can pick which file to open.
Then the script will loop through all available fields and plot the ones a user requests.

SCRIPTS FOR REAL DATA

real_surf.gs Plot some surface data
Need input data on model levels

plevels.gs Plot some pressure level fields
Need model output on pressure levels

rain.gs Plot total rainfall
Need a model output data set (any vertical coordinate), that contain fields "RAINC" and "RAINNC"

cross_z.gs Need z level data as input
Will plot a NS and EW cross section of RH and T (C)
Plots will run through middle of the domain

zlevels.gs Plot some height level fields
Need input data on height levels
Will plot data on 2, 5, 10 and 16km levels

input.gs Need WRF INPUT data on height levels

SCRIPTS FOR IDEALIZED DATA

bwave.gs Need height level data as input
Will look for 0.25 and 2 km data to plot

grav2d.gs Need normal model level data

hill2d.gs Need normal model level data

qss.gs Need height level data as input.
Will look for heights 0.75, 1.5, 4 and 8 km to plot

sqx.gs Need normal model level data a input

sqy.gs Need normal model level data a input

Examples of plots created for both idealized and real cases are available from:
http://www.mmm.ucar.edu/wrf/users/graphics/ARWpost/ARWpost.htm

Trouble Shooting

*The code executes correctly, but you get "**NaN**" or "**Undefined Grid**" for all fields when displaying the data.*

Look in the .ctl file.

a) If the second line is:

> **options byteswapped**
>
> Remove this line from your .ctl file and try to display the data again.
> If this SOLVES the problem, you need to remove the **-Dbytesw** option from configure.arwp

b) If the line below does NOT appear in your .ctl file:

> **options byteswapped**
>
> ADD this line as the second line in the .ctl file.
> Try to display the data again.
> If this SOLVES the problem, you need to ADD the **-Dbytesw** option for configure.arwp

The line "options byteswapped" is often needed on some computers (DEC alpha as an example). It is also often needed if you run the converter on one computer and use another to display the data.

Vis5D

For general information about working with Vis5D, view the Vis5D home page: *http://www.ssec.wisc.edu/~billh/vis5d.html*

WPP

The NCEP WRF Postprocessor was designed to interpolate both WRF-ARW and WRF-NMM 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) GRIB1 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).

This document will mainly deal with running the WPP package for the WRF-ARW modeling system. For details on running the package for the WRF-NMM system, please refer to the WRF-NMM User's Guide *(http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/WPS/index.php)*.

Necessary software

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/home.html*) or GEMPAK (*http://my.unidata.ucar.edu/content/software/gempak/index.html*). These packages are not part of the WPP installation and would need to be installed.

The WRF Postprocessor 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 -xvf wrfpostproc_v3.0.tar.gz
```

This command will create a directory called WPPV3. Under the main directory, there are five subdirectories:

- *sorc/*, 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.
- *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 the ***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.

Building the WPP Code

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

```
./configure

Please select from the following supported platforms.
1. LINUX (PC)
2. AIX (IBM)

Enter selection [1-2]:        1
Enter your NETCDF path:       /usr/local/netcdf-pgi
Enter your WRF model source code path:  /home/user/WRFV3

"YOU HAVE SELECTED YOUR PLATFORM TO BE:" LINUX
```

To modify the default compiler options, edit the appropriate platform specific makefile *(i.e. makefile_linux or makefile_ibm)* and repeat the configure process.

From the **WPPV3** directory, type:

```
make  >& 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)***.

Note: The ***makefile*** included in the tar file currently only contains the setup for single processor compilation of ***wrfpost*** for LINUX. Those users wanting to implement the parallel capability of this portion of the package will need to modify the compile options for ***wrfpost*** in the ***makefile***.

WPP Functionalities

The WRF Postprocessor v3.0,

- is compatible with WRF version 2.2 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, ***wrfpost*** and ***copygb***:

wrfpost

- Interpolates the forecasts from the model's native vertical coordinate to NWS standard output levels (*pressure, height, etc.*) and computes mean sea level pressure. If the requested field is on a model's native level, then no vertical interpolation is performed.
- Computes diagnostic output quantities.
  A list of available fields is shown in *Table 1*.
- Outputs the results in NWS and WMO standard GRIB1 format (for GRIB documentation, see *http://www.nco.ncep.noaa.gov/pmb/docs/*).
- De-staggers the WRF-ARW forecasts from a C-grid to an A-grid.
- Outputs two navigation files, **copygb_nav.txt** and **copygb_hwrf.txt** (these are ONLY used for WRF-NMM).

copygb

- Since *wrfpost* de-staggers WRF-ARW from a C-grid to an A-grid, WRF-ARW data can be displayed directly without going through *copygb*.
- No de-staggering is applied when posting WRF-NMM forecasts. Therefore, the posted WRF-NMM output is still on the staggered native E-grid and must go through *copygb* to be interpolated to a regular non-staggered grid.
- copygb is mainly used by WRF-NMM - see the WRF-NMM User's Guide (*http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/WPS/index.php*).

An additional utility called ***ndate*** is distributed with the WRF Postprocessor tar-file. This utility is used to format the dates of the forecasts to be posted for ingestion by the codes.

Computational Aspects and Supported Platforms

The WRF Postprocessor v3.0 has been tested on IBM and LINUX platforms. For LINUX, the Portland Group (PG) compiler has been used.

Only ***wrfpost*** has been parallelized, because it requires several 3-dimensional arrays 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.

Setting up the WRF model to interface with the WRF Postprocessor

The ***wrfpost*** program is currently set up to read a large number of fields from the WRF model history (*wrfout*) files. This configuration stems from NCEP's need to generate all of its required operational products. The program is configured such that is 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 diagnostical purposes by the WPP package. For example, if fields on isobaric surfaces are requested, but the pressure fields on model surfaces (*PB and P*) 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.

Note: For WRF-ARW, the accumulated precipitation fields (***RAINC*** and ***RAINNC***) are run total accumulations, whereas the WRF-NMM accumulated precipitation fields (***CUPREC*** and ***ACPREC***) are zeroed every 6 hours.

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, for example:

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

where,

- The first line specifies the field (*e.g. PRESS*) to process, the level type a user is interested in (*e.g. ON MDL SFCS*), and the degree of accuracy to be retained in the GRIB output (*SCAL=3.0*).
  A list of all possible output fields for ***wrfpost*** is provided in Table 1. 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 processed.

Controlling which fields *wrfpost* outputs

To output a field, the body of the control file needs to contain an entry for the appropriate field and output for this field 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 1000mb)*. The complete list of levels is specified in *sorc/wrfpost/POSTDATA.f*
- 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 *wrf_cntrl.param* so that 6 output levels are processed.
- 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 can 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

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

1. *cd* to your *DOMAINPATH* directory.

2. Make the following directories. The first will hold the WRF Postprocessor results. The second is where you will place your copy of the *wrf_cntrl.parm* file.

```
mkdir postprd
mkdir  parm
```

3. Copy the default *WPPV3/parm/wrf_cntrl.parm* to your working. Edit the *wrf_cntrl.parm* file to reflect the fields and levels you want *wrfpost* to output.

4. Copy the script *(WPPV3/scripts/run_wrfpost*)* of your choice to the *postprd*/.

5. 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 WPP run scripts

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 the location of the post-processor executables.

5. Link the microphysical table *${WRFPATH}/run/ETAMP_DATA* and the 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 .. d*nn*"

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:

   mpirun -np N wrfpost.exe < *itag* > *outpost* (for LINUX-MPI systems)

mpirun.lsf wrfpost.exe < itag > outpost (for IBM)

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* will generate the output file ***WRFPRS_dnn.hh*** *(linked to wrfpr_dnn.hh)*, in the post-processor working directory, where *"nn"* is the domain ID and *"hh"* the forecast hour. In addition, the script *run_wrfpostandgrads* will produce a suite of gif images named *variablehh_dnn_GrADS.gif*, and the script *run_wrfpostandgempak* will produce a suite of gif images named *variable_dnn_hh.gif*.

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.

Visualization

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 WPP 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* 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://my.unidata.ucar.edu/content/software/gempak/index.html

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
```

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 Users' Guide at: http://grads.iges.org/grads/gadoc/.
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 WPP, can be used to run *wrfpost* 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:

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

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

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

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

2. Link script *cbar.gs* to the post-processor working directory. *(This script is provided in WPP, and the run_wrfpostandgrads script makes a link from scripts/ to postprd/.)* To generate the above plots, 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*

Fields produced by the WRF Postprocessor

Table 1 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 1: *Fields produced by **wrfpost** (column 1), abbreviated names used in **wrfpost** control 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 longwave radiative flux	LW RAD TEMP TNDY	251	109
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	FRICTION 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

VAPOR

VAPOR is the **V**isualization and **A**nalysis **P**latform for **O**cean, Atmosphere, and Solar **R**esearchers. VAPOR was developed at NCAR to provide interactive visualization and analysis of numerically simulated fluid dynamics. With the latest (1.2) version, VAPOR now supports visualization of WRF-ARW simulation output.

Basic capabilities of VAPOR with WRF-ARW output

- *Direct Volume rendering (DVR)*
  Any 3D variable in the WRF data can be viewed as a density. Users control transparency and color to view temperature, water vapor, clouds, etc. in 3D.

- *Flow*
  - Draw 2D and 3D streamlines and flow arrows, showing the wind motion and direction, and how wind changes in time.
  - Path tracing (unsteady flow) enables visualization of trajectories that particles take over time. Users control when and where the particles are released.

- *Isosurfaces*
  The isosurfaces of variables are displayed interactively. Users can control iso-values, color and transparency of the isosurfaces.

- *Contour planes and Probes*
  3D variables can be intersected with arbitrarily oriented planes. Contour planes can be interactively positioned. Users can interactively pinpoint the values of a variable and establish seed points for flow integration.

- *Animation*
  Control the time-stepping of the data, for interactive replaying and for recording animated sequences.

- *Terrain rendering*
  The ground surface can be represented as a colored surface or can display a terrain image for geo-referencing.

VAPOR requirements

VAPOR is supported on Linux, Mac, Irix, and Windows. VAPOR works best with a recent graphics card (say 1-2 years old). The advanced features of VAPOR perform best with nVidia or ATI graphics accelerators.

VAPOR is installed on NCAR visualization systems. Users with UCAR accounts can connect their (windows or Linux) desktops to the NCAR visualization systems using NCAR's remote graphics service, and run VAPOR remotely. Instructions for using this are at: *http://www.cisl.ucar.edu/hss/dasg/services/docs/VAPOR.shtml*. Contact *dasg@ucar.edu* for assistance.

VAPOR support resources

The VAPOR website*:* *http://www.vapor.ucar.edu* includes software, documentation, example data, and links to other resources.

The VAPOR sourceforge website (*http://sourceforge.net/projects/vapor*) enables users to post bugs, request features, download software, etc.

Users of VAPOR on NCAR visualization systems should contact *dasg@ucar.edu* for support.

Questions, problems, bugs etc. should be reported to *vapor@ucar.edu*.

VAPOR development priorities are established by the VAPOR steering committee, a group of turbulence researchers who are interested in improving the ability to analyze and visualize time-varying simulation results. Post a feature request to the VAPOR sourceforge website, or e-mail *vapor@ucar.edu* if you have requests or suggestions about improving VAPOR capabilities.

How to use VAPOR with WRF-ARW data

1. Install VAPOR

VAPOR installers for Windows, Macintosh and Linux are available on the VAPOR download page, *http://www.vapor.ucar.edu/download*. You will be asked to agree to the terms of a BSD open source license. For most users, a binary installation is fine. Installation instructions are provided at the top of the VAPOR documentation page, *http://www.vapor.ucar.edu/doc*.

After VAPOR is installed, it is necessary to perform user environment setup on Unix or Mac, before executing any VAPOR software. These setup instructions are provided on the VAPOR binary install documentation page, *http://www.vapor.ucar.edu/doc/binary-install/index.shtml*.

2. Convert WRF output data to VAPOR

VAPOR datasets consist of (1) a metadata file (file type .vdf) that describes an entire VAPOR data collection, and (2) a directory of multi-resolution data files where the

actual data is stored. The metadata file is created by the command *wrfvdfcreate*, and the multi-resolution data files are written by the command *wrf2vdf*. The simplest way to create a VAPOR data collection is as follows:

First issue the command:

```
wrfvdfcreate wrf_files metadata_file.vdf
```

where: *wrf_files* is a list of one or more wrf output files that you want to use. *metadata_file.vdf* is the name that you will use for your metadata file.

For example:

```
wrfvdfcreate wrfout_d02_2006-10-25_18_00_00 wrfout.vdf
```

Then issue the command:

```
wrf2vdf metadata_file.vdf wrf_files
```

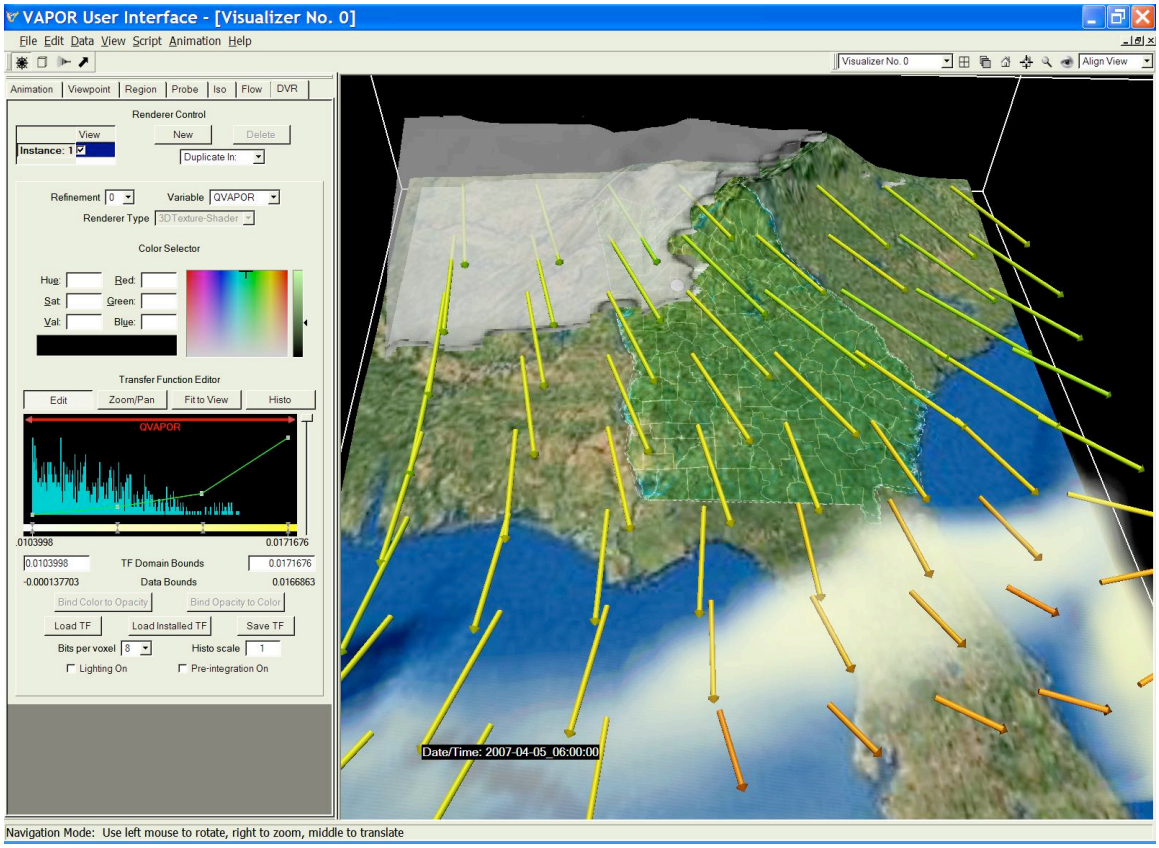
using the same arguments (in reversed order) as you used with *wrfvdfcreate*. Note that *wrf2vdf* does most of the work, and may take a few minutes to convert a large WRF dataset.

After issuing the above commands, all of the 3D variables in the specified WRF output files will be converted, for all the time steps in the files. If you desire more control over the conversion process, there are many additional options that you can provide to *wrfvdfcreate* and *wrf2vdf*. Type the command with the argument "-help" to get a short-listing of the command usage. All data conversion options are detailed in section 1 of *http://www.vapor.ucar.edu/doc/WRFsupport.pdf*. Some of the options include:

- Calculation of derived variables such as vertical vorticity, temperature in Kelvin, normalized pressure, wind velocity.
- Overriding default volume dimensions and/or spatial extents.
- Converting only a subset of the WRF output time steps
- Converting a specific collection of variables.

3. Visualize the WRF data

From the command line, issue the command "vaporgui", or double-click the VAPOR desktop icon (on Windows or Mac). This will launch the VAPOR user interface. From the Data menu, choose "Load a dataset into default session", and select the metadata file that you associated with your converted WRF data.



To visualize the data, select a renderer tab (DVR, Iso, Flow, or Probe), and then, at the top of that tab, check the box labeled "Instance:1", to enable that renderer. For example, the above image combines volume, flow and isosurface visualization with a terrain image.

There are many capabilities in VAPOR to support visualization of WRF data. Several resources are available to help users quickly get the information they need to obtain the most useful visualizations:

-The *Georgia Weather Case Study* (http://www.vapor.ucar.edu/doc/GeorgiaCaseStudy.pdf) provides a step-by-step tutorial, showing how to use most of the VAPOR features that are useful in WRF visualization.

- To understand the meaning or function of an element in the VAPOR user interface:

Tool tips: Place the cursor over a widget for a couple of seconds and a one-sentence description is provided.

Context-sensitive help: From the Help menu, click on "?Explain This", and then click with the left mouse button on a widget, to get a longer technical explanation of the functionality.

- Complete documentation of all capabilities of the VAPOR user interface is provided in the *VAPOR User Interface Reference Manual* (*http://www.vapor.ucar.edu/doc/ReferenceManual.pdf*).

- The *VAPOR Quick Start Guide* (*http://www.vapor.ucar.edu/doc/QuickstartGuide.pdf*) provides a step-by-step tutorial for using VAPOR on turbulence data. The Quick Start Guide does not discuss the WRF-specific capabilities of VAPOR.

- The WRF-specific features of VAPOR are described in detail in section 2 of the document "*Vapor Support for converting and visualizing WRF datasets*" (*http://www.vapor.ucar.edu/doc/WRFsupport.pdf*).

Utility: read_wrf_nc

This utility allows a user to look at a WRF netCDF file at a glance.

What is the difference between this utility and the netCDF utility ncdump?

- This utility has a large number of options, to allow a user to look at the specific part of the netCDF file in question.
- The utility is written in Fortran 90, which will allow users to add options.
-

This utility can be used for both WRF-ARW and WRF-NMM cores.
It can be used for geogrid, metgrid and wrf input / output files.
Only 3 basic diagnostics are available, pressure / height / tk, these can be activated with the -diag option *(these are only available for wrfout files)*

Obtain the **read_wrf_nc utility** from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)

Compile

The code should run on any machine with a netCDF library *(If you port the code to a different machine, please forward the compile flags to wrfhelp@ucar.edu)*

To compile the code, use the compile flags at the top of the utility.

e.g., *for a LINUX machine you need to type:*

```
pgf90  read_wrf_nc.f  -L/usr/local/netcdf/lib
   -lnetcdf  -lm  -I/usr/local/netcdf/include
   -Mfree  -o read_wrf_nc
```

If successful, this will create the executable: `read_wrf_nc`

Run

```
./read_wrf_nc  wrf_data_file_name  [-options]

options : [-h / help] [-att] [-m] [-M z] [-s]
          [-S x y z] [-v VAR] [-V VAR] [-w VAR]
          [-t t1 [t2]] [-times]
          [-ts xy X  Y  VAR VAR ....]
          [-ts ll  lat lon VAR VAR ....]
          [-lev z] [-rot] [-diag]
          [-EditData VAR]
```

Options:	*(Note: options [-att] ; [-t] and [-diag] can be used with other options)*
-h / help	Print help information.
-att	Print global attributes.
-m	Print list of fields available for each time, plus the min and max values for each field.
-M **z**	Print list of fields available for each time, plus the min and max values for each field. The min and max values of 3d fields will be for the **z** level of the field.
-s	Print list of fields available for each time, plus a sample value for each field. Sample value is taken from the middle of model domain.
-S **x y z**	Print list of fields available for each time, plus a sample value for each field. Sample value is at point **x y z** in the model domain.
-t **t1 [t2]**	Apply options only to times **t1** to **t2**. t2 is optional. If not set, options will only apply to **t1**.
-times	Print only the times in the file.
-ts	Generate time series output. A full vertical profile for each variable will be created. -ts xy X Y VAR VAR ….. will generate time series output for all VAR's at location X/Y -ts ll lat lon VAR VAR ….. will generate time series output for all VAR's at x/y location nearest to lat/lon
-lev **z**	Work only with option **–ts** Will only create a time series for level **z**
-rot	Work only with option **–ts** Will rotate winds to earth coordinates
-diag	Add if you want to see output for the diagnostics temperature (K), full model pressure and model height *(tk, pressure, height)*
-v **VAR**	Print basic information about field **VAR**.
-V **VAR**	Print basic information about field **VAR**, and dump the full field out to the screen.
-w **VAR**	Write the full field out to a file **VAR.out**
	Default Options are [-att –s]

SPECIAL option: -EditData VAR

This option allows a user to **read** a WRF netCDF file, **change** a specific field and **write** it BACK into the WRF netCDF file.
This option will CHANGE your CURRENT WRF netCDF file so TAKE CARE when using this option.
ONLY one field at a time can be changed. So if you need 3 fields changed, you will need to run this program 3 times, each with a different "VAR"
IF you have multiple times in your WRF netCDF file – **by default ALL times** for variable "VAR" WILL be changed. *If you only want to change one time period, also use the "-t" option.*

HOW TO USE THIS OPTION:

Make a COPY of your WRF netCDF file before using this option

EDIT the subroutine USER_CODE

ADD an IF-statement block for the variable you want to change. This is to prevent a variable getting overwritten by mistake.

For REAL data arrays, work with array "data_real" and for INTEGER data arrays, work with the array "data_int".

Example 1:
If you want to change all (all time periods too) values of U to a constant 10.0 m/s, you would **add** the following IF-statement:

```
  else if ( var == 'U') then
    data_real = 10.0
```

Example 2:
If you want to change a section of the LANDMASK data to SEA points:

```
  else if ( var == 'LANDMASK') then
    data_real(10:15,20:25,1) = 0
```

Example 3:
Change **all** ISLTYP category 3 values into category 7 values (NOTE this is an INTEGER field):

```
  else if ( var == 'ISLTYP') then
    where (data_int == 3 )
      data_int = 7
    end where
```

Compile and run program
You will be prompted if this is really what you want to do
ONLY the answer "yes" will allow the change to take effect

Utility: iowrf

This utility allows a user to do some basic manipulation on WRF-ARW netCDF files.

- The utility allows a user to thin the data; de-stagger the data; or extract a box from the data file.

Obtain the **iowrf utility** from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)

Compile

The code should run on any machine with a netCDF *library (If you port the code to a different machine, please forward the compile flags to wrfhelp@ucar.edu)*

To compile the code, use the compile flags at the top of the utility.

e.g., *for a LINUX machine you need to type:*

```
pgf90 iowrf.f  -L/usr/local/netcdf/lib  -lnetcdf  -lm
      -I/usr/local/netcdf/include  -Mfree  -o iowrf
```

If successful, this will create the executable: `iowrf`

Run

```
./iowrf  wrf_data_file_name  [-options]

options : [-h / help] [-thina X] [-thin X] [-box {}]
          [-A] [-64bit]
```

-thina **X**	Thin the data with a ratio of 1:X Data will be averaged before being fed back
-thin **X**	Thin the data with a ratio of 1:X No averaging will be done
-box **{}**	Extract a box from the data file. X/Y/Z can be controlled independently. e.g., -box **x** 10 30 **y** 10 30 **z** 5 15 -box **x** 10 30 **z** 5 15 -box **y** 10 30 -box **z** 5 15
-A	De-stagger the data – no thinning will take place
-64bit	Allow large files (> 2GB) to be read / write

Utility: p_interp

This utility interpolates WRF-ARW netCDF output files to user specified pressure levels.

Obtain the **p_interp utility** from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)

Compile

The code should run on any machine with a netCDF *library (If you port the code to a different machine, please forward the compile flags to wrfhelp@ucar.edu)*

To compile the code, use the compile flags at the top of the utility.

e.g., *for a LINUX machine you need to type:*

```
pgf90 p_interp.F90  -L/usr/local/netcdf/lib
      -lnetcdf  -lm  -I/usr/local/netcdf/include
      -Mfree  -o p_interp
```

If successful, this will create the executable: `p_interp`

Run

Edit the associated namelist.pinterp file *(see namelist options below),* and run

```
./p_interp
```

&io	
input_root_name	Path and file name(s) of wrfout files. *Use wild character if more than one file is processed.* *Output will be written to input_root_name_PLEV.*
process	Indicate which fields to process. 'all' fields in wrfout file *(diagnostics PRES, TT & GEOPT will automatically be calculated)*; 'list' of fields as indicated in 'fields'
fields	List of fields to process.
debug	Switch debug more on/off.
-64bit	Allow large files (> 2GB) to be read / write.

&interp_in	
interp_levels	List of pressure levels to interpolate data to
extrapolate	0 - set values below ground and above model top to missing values *(default)* 1 - extrapolate below ground, and set above model top to model top values
interp_method	1 - linear in p interpolation *(default)* 2 - linear in log p interpolation
unstagger_grid	Set to .True. so unstagger the data on output

Tools

Below is a list of tools that are freely available that can be used very successfully to manipulate model data (both WRF model data as well as other GRIB and netCDF datasets).

Converting Graphics

ImageMagick

ImageMagick is a software suite to create, edit, and compose bitmap images. It can read, convert and write images in a variety of formats (over 100) including DPX, EXR, GIF, JPEG, JPEG-2000, PDF, PhotoCD, PNG, Postscript, SVG, and TIFF. Use ImageMagick to translate, flip, mirror, rotate, scale, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and B_zier curves.

The software package is freely available from, *http://www.imagemagick.org*. Download and installation instructions are also available from this site.

Examples of converting data with ImageMagick software:

```
convert  file.pdf     file.png
convert  file.png     file.bmp
convert  file.pdf     file.gif
convert  file.ras     file.png
```

ImageMagick cannot convert ncgm (NCAR Graphics) file format to other file formats.

Converting ncgm (NCAR Graphics) file format

NCAR Graphics has tools to convert ncgm files to raster file formats. Once files are in raster file format, ImageMagick can be used to translate the files into other formats.

For *ncgm* files containing a single frame, use ***ctrans***.

```
ctrans  -d sun  file.ncgm  file.ras
```

For *ncgm* files containing multiple frames, first use ***med*** (metafile frame editor) and then *ctrans*. ***med*** will create multiple single frame files called *medxxx.ncgm*

```
med -e '1,$ split $' file.ncgm
ctrans  -d sun_ med001.ncgm > med001.ras
```

Design WRF model domains

WPS/util/plotgrids.exe, can be used to display model domains before WPS/geogrid.exe is run.

This utility reads the domain setup from namelist.wps and creates an ncgm file that can be viewed with the NCAR Graphics command "idt", e.g.,

```
idt gmeta
```

Read more about this utility in Chapter 3 of this Users Guide.

Display ungrib (intermediate) files

WPS/util/plotfmt.exe, can be used to display intermediate files created by WPS/ungrib.exe.

If you have created intermediate files manually, it is a very good practice to use this utility to display the data in your files first before running WPS/metgrid/exe.
***Note**: If you plan on manually creating intermediate files, refer to http://www.mmm.ucar.edu/wrf/OnLineTutorial/WPS/IM_files.htm for detailed information about the file formats and sample programs.*

This utility reads intermediate files and creates an ncgm file that can be viewed with the NCAR Graphics command "idt", e.g.,

```
idt gmeta
```

Read more about this utility in **Chapter 3** of this Users Guide.

netCDF data

netCDF stands for **ne**twork **C**ommon **D**ata **F**orm.
Most of the information below can be used for WRF netCDF data as well as other netCDF datasets.
netCDF is one of the current supported data formats chosen for WRF I/O API.

Advantages of using netCDF?
Most graphical packages support netCDF file formats
netCDF files are platform-independent (big-endian / little-endian)
A lot of software already exists which can be used to process/manipulate netCDF data

Documentation:
http://www.unidata.ucar.edu/ (General netCDF documentation)
http://www.unidata.ucar.edu/software/netcdf/fguide.pdf (NETCDF User's Guide for FORTRAN)

Utilities:
ncdump
Part of the netCDF libraries. Reads a netCDF file and prints information about the dataset. e.g.

`ncdump -h` *`file`* *(print header information)*
`ncdump -v` *`VAR file`* *(print header information and the full field VAR)*
`ncdump -v Times file` *(a handy way to see how many times are available in a WRF output file)*

ncview
Display netCDF data graphically. No overlays, no maps and no manipulation of data possible.
http://meteora.ucsd.edu/~pierce/ncview_home_page.html

ncBrowse
Display netCDF data graphically. Some overlays, maps and manipulation of data are possible.
http://www.epic.noaa.gov/java/ncBrowse/

read_wrf_nc
A utility to display basic information about WRF netCDF files.

iowrf
A utility to do some basic file manipulation on WRF-ARW netCDF files.

p_interp
A utility to interpolate WRF-ARW netCDF output files to user specified pressure levels.

netCDF operators
http://nco.sourceforge.net/
Stand alone programs to, which can be used to manipulate data (performing grid point averaging / file differencing / file 'appending'). *Examples of the available operators are ncdiff, ncrcat, ncra, and ncks.*

ncdiff
Difference two file, e.g.

```
ncdiff  input1.nc input2.nc output.nc
```

ncrcat
Write specified variables / times to a new file, e.g.

```
ncrcat -v RAINNC wrfout* RAINNC.nc
ncrcat -d Time,0,231 -v  RAINNC  wrfout* RAINNC.nc
```

ncra
Average variables and write to a new file, e.g.

```
ncra -v OLR  wrfout* OLR.nc
```

ncks (nc kitchen sink)
Combination of NCO tools all in one (handy: one tool for multiple operations).

GRIB data

Documentation

http://dss.ucar.edu/docs/formats/grib/gribdoc/ (Guide to GRIB 1)
http://www.nco.ncep.noaa.gov/pmb/docs/grib2/grib2_doc.shtml (Guide to GRIB2)
http://www.nco.ncep.noaa.gov/pmb/docs/grib2/GRIB2_parmeter_conversion_table.html (GRIB2 - GRIB1 parameter conversion table)

GRIB codes

It is important to understand the GRIB codes to know which fields are available in your dataset. For instance, NCEP uses the GRIB1 code 33 for the U-component of the wind, and 34 for the V-component. *Other centers may use different codes, so always obtain the GRIB codes from the center you get your data from.*

GRIB2 uses 3 codes for each field - **product**, **category** and **parameter**.
We would most often be interested in **product 0** (*Meteorological products*).
Category refers to the type of field, e.g., category 0 is temperature, category 1 is moisture and category 2 is momentum. **Parameter** is the field number.
So whereas GRIB1 only uses code 33 for the U-component of the wind, GRIB2 will use 0,2,2, for the U-component, and 0,2,3 for the V-component.

Display GRIB header/field information

GRIB1 data

WPS/util/g1print.exe
wgrib *(http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html)*

GRIB2 data
WPS/util/g2print.exe
wgrib2 *(http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/)*

Convert GRIB1 data to netCDF format
ncl_grib2nc *(http://www.ncl.ucar.edu/Document/Tools)*

Model Verification

MET is designed to be a highly configurable, state-of-the-art suite of verification tools. It was developed using output from the Weather Research and Forecasting (WRF) modeling system but may be applied to the output of other modeling systems as well.

MET provides a variety of verification techniques, including:

- Standard verification scores comparing gridded model data to point-based observations
- Standard verification scores comparing gridded model data to gridded observations
- Object-based verification method comparing gridded model data to gridded observations

http://www.dtcenter.org/met/users/index.php