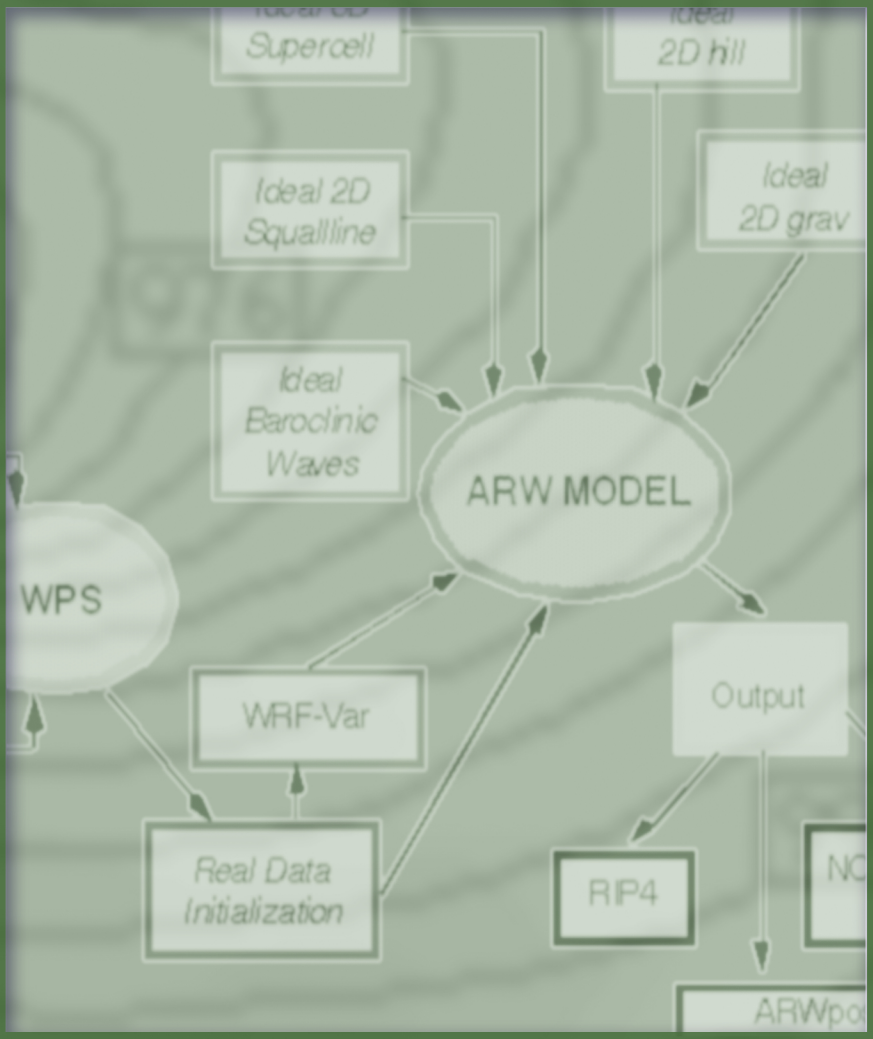
WEATHER RESEARCH & FORECASTING

ARW

Version 2 Modeling System User's Guide
January 2007



Mesoscale & Microscale Meteorology Division * National Center for Atmospheric Research

Foreword

This User's Guide describes the Advanced Research WRF (ARW) Version 2.2 modeling system, released in December 2006. As the ARW is developed further, this document will be continuously enhanced and updated.

This version contains the documentation for the new WRF Preprocessing System (WPS), which replaces the old WRFSI system. The documentation for the old WRFSI system is retained as Appendix A of this document. New documentation for the post-processing software ARWpost has been included. This package replaces WRF2GrADS and WRF2VIS5D. Documentation for these two programs is retained in Appendix B.

Highlights of changes to WRFV2 include:

- New physics options: CAM radiation, Thompson microphysics, and urban canopy model coupled with Noah;
- Positive-definite advection for moisture, scalars, tke and chemical variables;
- Sixth order horizontal numerical diffusion;
- Vertical interpolation in real to work with WPS data;
- FDDA analysis and observation nudging

For the latest version of this document, please visit the ARW User's 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
John Bray
Cindy Bruyère
Michael Duda
Jimy Dudhia
Dave Gill
John Michalakes

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2.2

1. Overview

- Introduction 1-1
- The WRF Modeling System Program Components 1-2

2. Software Installation

- Introduction 2-1
- Required Compilers and Scripting Languages 2-2
- Required/Optional Libraries to Download 2-2
- Post-Processing Utilities 2-3
- Unix Environment Settings 2-4
- Building the WRF Code 2-4
- Building the WRF VAR Code 2-5
- Building the WPS Code 2-5

3. The WRF Preprocessing System (WPS)

- Introduction 3-1
- Function of Each WPS Program 3-2
- Installing the WPS 3-4
- Running the WPS 3-7
- Creating Nested Domains with the WPS 3-14
- Using Multiple Data Sources 3-17
- Parallelism in the WPS 3-19
- Checking WPS Output 3-20
- WPS Utility Programs 3-21
- Writing Meteorological Data to the Intermediate Format 3-23
- Creating and Editing Vtables 3-25
- Writing Static Data to the Geogrid Binary Format 3-27
- Description of Namelist Variables 3-30
- Description of GEOGRID.TBL Options 3-35
- Description of METGRID.TBL Options 3-40

4. WRF Initialization

- Introduction 4-1
- Initialization for Ideal Data Cases 4-2
- Initialization for Real Data Cases 4-4

5. WRF Model

- Introduction 5-1
- Software Requirement 5-2
- Before You Start 5-2
- How to Compile WRF? 5-3
- How to Run WRF? 5-6
- Check Output 5-18
- Physics and Diffusion Options..... 5-19
- Description of Namelist Variables 5-22
- List of Fields in WRF Output 5-37

6. WRF_VAR

- Introduction 6-1
- Goals Of This WRF-Var Tutorial 6-2
- Tutorial Schedule 6-4
- Download Test Data 6-4
- The 3D-Var Observation Preprocessor (3DVAR_OBSPROC) 6-6
- Setting up WRF-Var 6-10
- Run WRF-Var CONUS Case Study 6-14
- WRF-Var Diagnostics..... 6-16
- Updating WRF lateral boundary conditions..... 6-19

7. WRF Software

- Introduction 7-1
- WRF Build Mechanism..... 7-1
- Registry..... 7-4
- I/O Applications Program Interface (I/O API)..... 7-5
- Timekeeping..... 7-6
- Software Documentation..... 7-7
- Portability and Performance..... 7-7

8. Post-Processing Programs

- Introduction 8-1
- NCL.. 8-2
- RIP4. 8-7
- ARWpost..... 8-13
- read_wrf_nc utility 8-19
- iowrf utility..... 8-22

Appendix A: WRF Standard Initialization

Appendix B: Old Post-Processing Programs

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 1: Overview

Table of Contents

- Introduction
- The WRF ARW Modeling System Program Components

Introduction

The Advanced Research WRF (ARW) modeling system has been in development for the past few years. The current release is Version 2. 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 2) 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

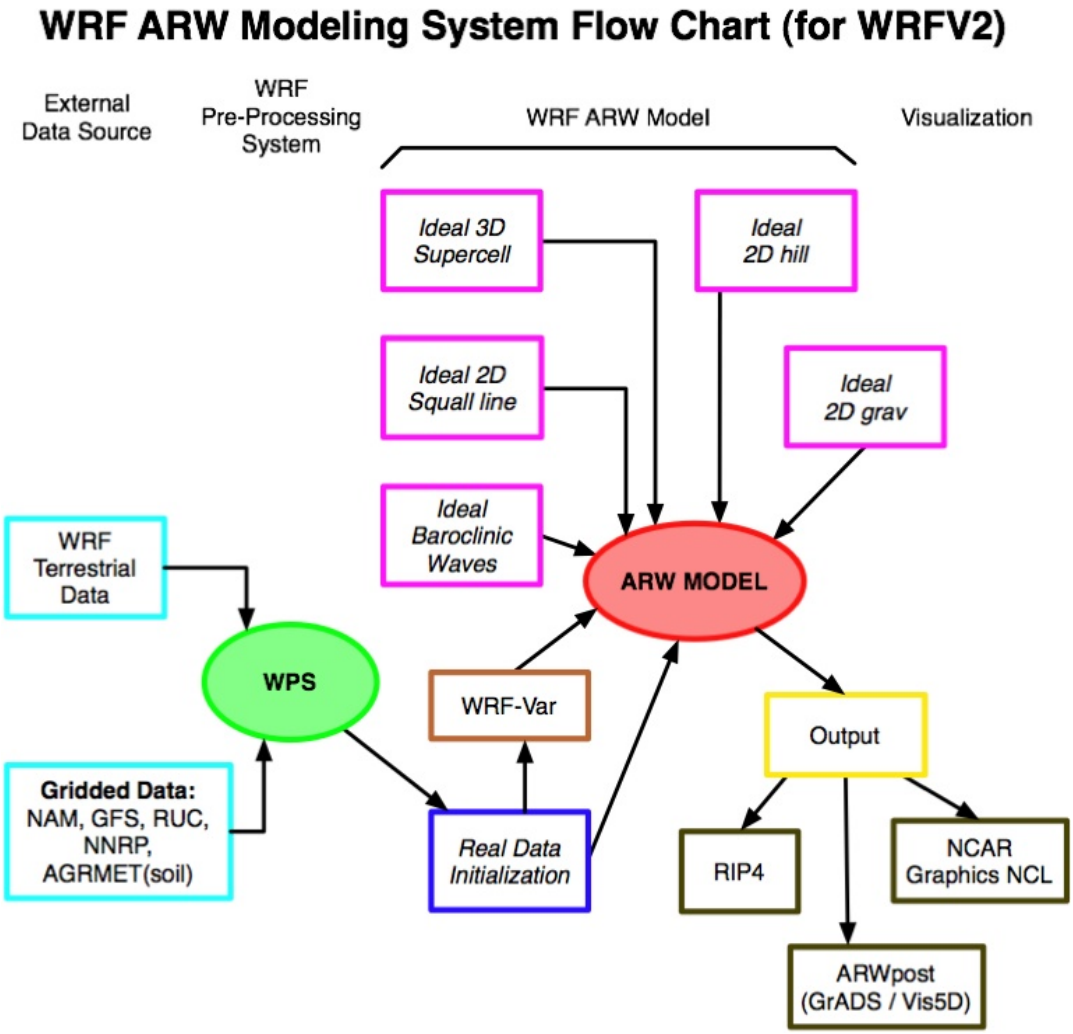
And these are the subjects of this document.

Other components of the WRF system will be supported for community use in the future, depending on interest and available resources.

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 2.



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 graphics 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.

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.

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 pragram. 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
- complete coriolis and curvature terms
- two-way nesting with multiple nests and nest levels
- one-way nesting
- moving nests
- mass-based terrain following coordinate (note that the height-based dynamic core is no longer supported)
- vertical grid-spacing can vary with height
- map-scale factors for conformal projections:
    - polar stereographic
    - Lambert-conformal
    - Mercator
- Arakawa C-grid staggering
- Runge-Kutta 2nd and 3rd order timestep 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
- lateral boundary conditions
    - idealized cases: periodic, symmetric, and open radiative
    - real cases: specified with relaxation zone
- full physics options for land-surface, PBL, radiation, microphysics and cumulus parameterization

- grid analysis nudging and observation nudging

Graphics 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.

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

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 2: Software Installation

Table of Contents

- Introduction
- Required Compilers and Scripting Languages
- Required/Optional Libraries to Download
- Post-Processing Utilities
- UNIX Environment Settings
- Building the WRF Code
- Building the WRF-Var Code
- Building the WPS Code

Introduction

The WRF modeling system software installation is fairly straightforward on the ported platforms. The package is mostly self-contained, meaning that WRF requires no external libraries (such as for FFTs or various linear algebra solvers). The one external package it does 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).

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
HP/Compaq	alpha	Tru64	vendor
HP/Compaq	IA64 (Intel)	Linux	vendor
HP/Compaq	IA64	HPUX	vendor
IBM	Power Series	AIX	vendor

SGI	IA64	Linux	Intel
SGI	MIPS	Irix	vendor
Sun	UltraSPARC	SunOS	vendor
COTS*	IA32/AMD 32	Linux	Intel / PGI
COTS	IA64/Opteron	Linux	Intel / PGI
Mac	G5	Darwin	xlf

* Commercial off the shelf systems

The WRF code runs on single processor machines, shared-memory machines (that use the OpenMP API), distributed memory machines (with the appropriate MPI libraries), and on distributed clusters (utilizing both OpenMP and MPI). The WRF 3DVAR code runs on most systems listed above too. The porting to systems that use the Intel compiler is currently under development. The Mac architecture is only supported as a serial build.

The WRFSI code also runs on most systems list above. Sun and Intel compiles are not yet supported.

Required Compilers and Scripting Languages

The WRF model (and WRF 3DVAR) is written in Fortran (what many refer to as Fortran 90). The software layer, RSL and now RSL_LITE, which sits between WRF and the MPI interface is written in C. There are also ancillary programs that are written in C to perform file parsing and file construction, both of which are required for default building of the WRF modeling code. Additionally, the WRF build mechanism uses several scripting languages: including perl (to handle various tasks such as the code browser designed by Brian Fiedler), 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.

The WRFSI is mostly written in Fortran 77 and Fortran 90 with a few C routines. Perl scripts are used to run the programs, and Perl/Tk is used for GUI.

Unix make is used in building all executables.

Required/Optional Libraries to Download

The only library that is *almost always* required is the netCDF package from Unidata (login > Downloads > NetCDF). Some of the WRF post-processing packages assume that

the data from the WRF model is using the netCDF libraries. One may also need to add /path-to-netcdf/netcdf/bin to your path so that one may execute netcdf command, such as **ncdump** and **ncgen**.

Hint: If one wants to compile WRF codes on a Linux system using PGI (Intel) compiler, make sure the netCDF library is installed using PGI (Intel) compiler, too.

There are optional external libraries that may be used within the WRF system: ESMF and PHDF. Neither the ESMF nor the PHDF libraries are required for standard builds of the WRF system.

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. 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 for GriB1 data processing, Todd Hutchinson (WSI) has provided a complete source library that is included with the software release.

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 data to RIP input format
    - table driven

UNIX Environment Settings

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

- `setenv WRF_EM_CORE 1`

  explicitly defines which model core to build

- `unset limits`
    - especially if you are on a small system
- `setenv MP_STACK_SIZE 64000000`
    - OpenMP blows through the stack size, set it large
- `setenv NETCDF /usr/local/netcdf` (or where ever you have it stuck)
    - WRF wants both the lib and the include directories
- `setenv MPICH_F90 f90` (or whatever your Fortran compiler may be called)
    - WRF needs the bin, lib, and 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

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 present 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).

- Get the WRF zipped tar file
    - WRFV2 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 WRFV2.2.TAR.gz | tar -xf -`

    - again, if there is a later version of the code grab it, 2.1 is just used as an example
- `cd WRFV2`
- `./configure`
    - choose one of the options
    - usually, option "1" is for a serial build, that is the best for an initial test
- `./compile em_real` (or any of the directory names in ./WRFV2/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**

Building the WRF-Var Code

See details in Chapter 6.

Building the WPS Code

WPS replaces the old WRFSI code.

Building WPS requires that WRFV2 is already build.

- Get the WPS zipped tar file
    - WPSV2.2.TAR.gz from http://www.mmm.ucar.edu/wrf/users/get_source.html
- unzip and untar the file
    - `gzip -cd WPSV2.2.TAR.gz | tar -xf -`
- `cd WPS`
- `./configure`
    - choose one of the options
    - usually, option "1" is for a serial build, that is the best for an initial test
- `./compile`
- `ls -ls *.exe`
    - you should see **geogrid.exe**, **ungrib.exe**, and **metgrid.exe**
    - if you built an ideal-data case, you should see **ideal.exe** and **wrf.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**, **plotgrid.exe**, and **rd_intermediate.exe**

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

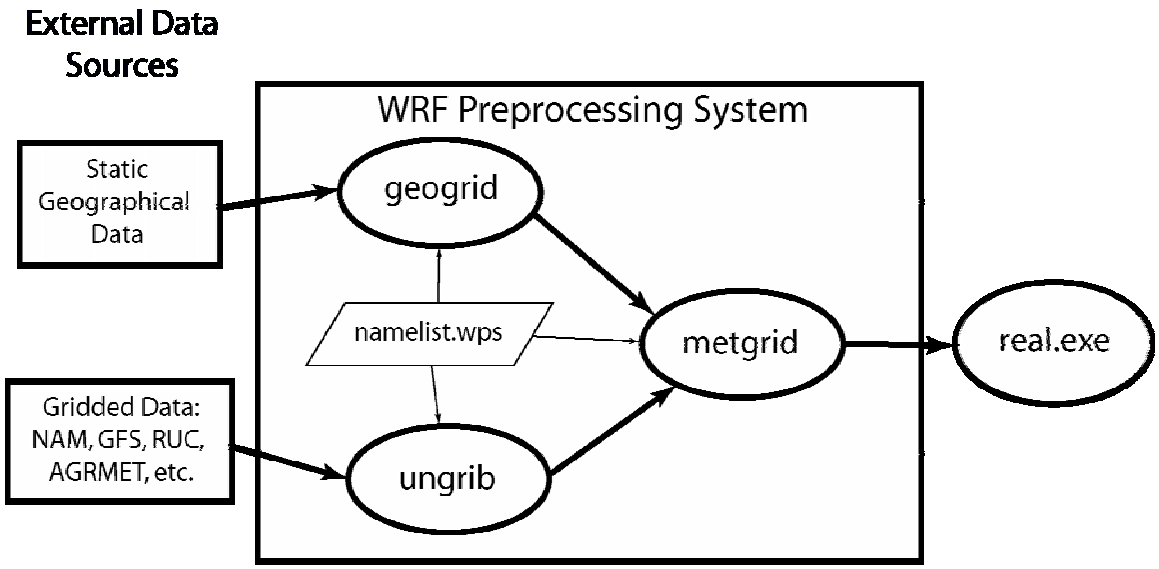
Chapter 3: The WRF Preprocessing System (WPS) – Preparing Input Data

Table of Contents

- Introduction
- Function of Each WPS Program
- Installing the WPS
- Running the WPS
- Creating Nested Domains with the WPS
- Using Multiple Data Sources
- Parallelism in the WPS
- Checking WPS Output
- WPS Utility Programs
- Writing Meteorological Data to the Intermediate Format
- Creating and Editing Vtables
- Writing Static Data to the Geogrid Binary Format
- Description of Namelist Variables
- Description of GEOGRID.TBL Options
- Description of METGRID.TBL Options

Introduction

The WRF Preprocessing System (WPS) is a set of three programs whose collective role is to prepare input to the real.exe 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.exe program, a task that was previously 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 purpose, contents, and format of these tables are documented elsewhere, and 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 larger 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 domain is 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 latitude and longitudes for 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 MMM website, 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'. 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 source 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 and 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 filesystem 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 available 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), and AFWA's AGRMET land surface model output. 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 real.exe 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 to be used for masked interpolations, and the staggering (e.g., U, V in ARW; H, V in NMM) to which a field is to be 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 I/O API; consequently, WRF must be compiled prior to installation of the WPS so that the I/O API libraries in the "external" directory of WRF 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.

Where WRF adds a software layer between the model and the communications package, the WPS parallel programs 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.

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
```

2) zlib (another compression library, which is 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
```

3) 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
```

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 used to build the WRF model. The following table lists the currently supported operating system and compiler combinations for WPS.

Architecture	OS	Compiler
IBM POWER-4/5	**AIX**	**xlf90**
Compaq Alpha	**OSF1**	**f90**
PC x86	**GNU/Linux 32-bit**	**PGI (pgf90)**
PC x86	**GNU/Linux 32-bit**	**Intel (ifort)**
PC x86	**GNU/Linux 32-bit**	**g95**
PC x86_64	**GNU/Linux 32-bit**	**PGI (pgf90)**
PC x86_64	**GNU/Linux 32-bit**	**PathScale (pathf90)**
SGI	**IRIX64**	**f90**
SGI Altix	**Linux**	**Intel (ifort)**
Sun	**SunOS**	**f90**
Apple G5	**OS X**	**xlf90**

Installation Steps

- Download the `WPS.tar.gz` file and unpack it at the same directory level as WRFV2, as shown below.

```
>ls

-rw-r--r--  1  537490 Oct 30 16:38 WPS.tar.gz
drwxr-xr-x 17     512 Oct 30 16:18 WRFV2

>tar xf WPS.tar.gz

>ls

drwxr-xr-x  8     512 Oct 30 16:38 WPS
-rw-r--r--  1  537490 Oct 30 16:38 WPS.tar.gz
drwxr-xr-x 17     512 Oct 30 16:18 WRFV2
```

- At this point, a listing of the current working directory should at least include the directories WRFV2 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     512 Oct 30 16:38 arch
-rwxr-xr-x 1    1672 Sep  8 18:50 clean
-rwxr-xr-x 1    3349 Sep 12 11:11 compile
-rw-r--r-- 1  100168 Nov  1 10:02 compile.output
-rwxr-xr-x 1    4257 Jul 19 13:47 configure
-rw-r--r-- 1    2465 Nov  1 10:00 configure.wps
drwxr-xr-x 5     512 Nov  1 10:02 geogrid
lrwxrwxrwx 1      23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
-rwxr-xr-x 1    1138 Aug  3 10:09 link_grib.csh
drwxr-xr-x 4     512 Nov  1 10:02 metgrid
lrwxrwxrwx 1      23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
-rw-r--r-- 1    1638 Oct 30 11:54 namelist.wps
-rw-r--r-- 1    5074 Sep 15 13:05 README
drwxr-xr-x 7     512 Oct 30 16:38 test_suite
drwxr-xr-x 4     512 Nov  1 10:02 ungrib
lrwxrwxrwx 1      21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
drwxr-xr-x 3     512 Nov  1 10:02 util
-rw-r--r-- 1      13 Oct 30 16:38 VERSION
```

Running the WPS

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

1. Define a model domain and nests with *geogrid*.
2. Extract meteorological data from GRIB data sets for the simulation period with *ungrib*.
3. Horizontally interpolate meteorological data 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. 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 of the WPS root directory should look something like:

```
      >ls

      drwxr-xr-x 2     512 Oct 30 16:38 arch
      -rwxr-xr-x 1    1672 Sep  8 18:50 clean
      -rwxr-xr-x 1    3349 Sep 12 11:11 compile
      -rw-r--r-- 1  100168 Nov  1 10:02 compile.output
      -rwxr-xr-x 1    4257 Jul 19 13:47 configure
      -rw-r--r-- 1    2465 Nov  1 10:00 configure.wps
      drwxr-xr-x 5     512 Nov  1 10:02 geogrid
      lrwxrwxrwx 1      23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
      -rwxr-xr-x 1    1138 Aug  3 10:09 link_grib.csh
      drwxr-xr-x 4     512 Nov  1 10:02 metgrid
      lrwxrwxrwx 1      23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
      -rw-r--r-- 1    1638 Oct 30 11:54 namelist.wps
      -rw-r--r-- 1    5074 Sep 15 13:05 README
      drwxr-xr-x 7     512 Oct 30 16:38 test_suite
      drwxr-xr-x 4     512 Nov  1 10:02 ungrib
      lrwxrwxrwx 1      21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
      drwxr-xr-x 3     512 Nov  1 10:02 util
      -rw-r--r-- 1      13 Oct 30 16:38 VERSION
```

The model domain and nests are defined in the “geogrid” namelist record of the namelist.wps file, and, in addition, 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 = '2006-08-16_12:00:00','2006-08-16_12:00:00',
 end_date   = '2006-08-16_18:00:00','2006-08-16_12:00:00',
 interval_seconds = 21600
 io_form_geogrid = 2,
 opt_output_from_geogrid_path = './',
 debug_level = 0
/

&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 = '/data3a/mp/gill/DATA/GEOG'
 opt_geogrid_tbl_path = 'geogrid/'
/
```

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`, and the number of nests, including the mother domain, must be chosen with `max_dom`. Additionally, a location (if not the default – the current working directory) where domain files should be written to may be chosen with `opt_output_from_geogrid_path`, 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. Again, the user is referred to a description of the namelist variables for more details on the meaning and possible values for each variable.

Having suitably defined the simulation coarse domain and nested domains, the geogrid.exe executable may be run to produce domain files, which are named `geo_em.d0N.nc`, where `N` is the number of the nest defined in each file. It is important, to note that the `geo_em` prefix is used for ARW domain files, while the `geo_nmm` prefix is used for NMM domain files; also, the file suffix will vary depending on the `io_form_geogrid` that is selected. To run geogrid, issue the following command:

```
>geogrid.exe
```

After running geogrid.exe, the success 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 './') 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     512 Oct 30 16:38 arch
-rwxr-xr-x 1    1672 Sep  8 18:50 clean
-rwxr-xr-x 1    3349 Sep 12 11:11 compile
-rw-r--r-- 1  100168 Nov  1 10:02 compile.output
-rwxr-xr-x 1    4257 Jul 19 13:47 configure
-rw-r--r-- 1    2465 Nov  1 10:00 configure.wps
```

```
      -rw-r--r-- 1 1831720 Nov  1 10:22 geo_em.d01.nc
      -rw-r--r-- 1 4443720 Nov  1 10:22 geo_em.d02.nc
      drwxr-xr-x 5     512 Nov  1 10:02 geogrid
      lrwxrwxrwx 1      23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
      -rw-r--r-- 1    9672 Nov  1 10:22 geogrid.log
      -rwxr-xr-x 1    1138 Aug  3 10:09 link_grib.csh
      drwxr-xr-x 4     512 Nov  1 10:02 metgrid
      lrwxrwxrwx 1      23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
      -rw-r--r-- 1    1638 Oct 30 11:54 namelist.wps
      -rw-r--r-- 1    5074 Sep 15 13:05 README
      drwxr-xr-x 7     512 Oct 30 16:38 test_suite
      drwxr-xr-x 4     512 Nov  1 10:02 ungrib
      lrwxrwxrwx 1      21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
      drwxr-xr-x 3     512 Nov  1 10:02 util
      -rw-r--r-- 1      13 Oct 30 16:38 VERSION
```

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 = '2006-08-16_12:00:00','2006-08-16_12:00:00',
 end_date   = '2006-08-16_18:00:00','2006-08-16_12:00:00',
 interval_seconds = 21600
 io_form_geogrid = 2,
 opt_output_from_geogrid_path = './',
 debug_level = 0
/

&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 mother domain (`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 only variable, `out_format`, defines 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`.

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 may 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 /mmmtmp/mm5rt/data/gfs, the files could be linked with link_grib.csh as in the following commands:

```
>ls /mmmtmp/mm5rt/data/gfs

-rw-r--r-- 1  24008620 Nov  1 08:25 gfs_060816_12_00
-rw-r--r-- 1  27106796 Nov  1 08:28 gfs_060816_12_06

>link_grib.csh /mmmtmp/mm5rt/data/gfs/gfs_061101_12_*
```

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

```
>ls

drwxr-xr-x 2      512 Oct 30 16:38 arch
-rwxr-xr-x 1     1672 Sep  8 18:50 clean
-rwxr-xr-x 1     3349 Sep 12 11:11 compile
-rw-r--r-- 1   100168 Nov  1 10:02 compile.output
-rwxr-xr-x 1     4257 Jul 19 13:47 configure
-rw-r--r-- 1     2465 Nov  1 10:00 configure.wps
-rw-r--r-- 1  1831720 Nov  1 10:22 geo_em.d01.nc
-rw-r--r-- 1  4443720 Nov  1 10:22 geo_em.d02.nc
drwxr-xr-x 5      512 Nov  1 10:02 geogrid
lrwxrwxrwx 1       23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
-rw-r--r-- 1     9672 Nov  1 10:22 geogrid.log
lrwxrwxrwx 1       39 Nov  1 10:35 GRIBFILE.AAA ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_00
lrwxrwxrwx 1       39 Nov  1 10:35 GRIBFILE.AAB ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_06
-rwxr-xr-x 1     1138 Aug  3 10:09 link_grib.csh
drwxr-xr-x 4      512 Nov  1 10:02 metgrid
lrwxrwxrwx 1       23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
-rw-r--r-- 1     1638 Oct 30 11:54 namelist.wps
-rw-r--r-- 1     5074 Sep 15 13:05 README
drwxr-xr-x 7      512 Oct 30 16:38 test_suite
drwxr-xr-x 4      512 Nov  1 10:02 ungrib
lrwxrwxrwx 1       21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
drwxr-xr-x 3      512 Nov  1 10:02 util
-rw-r--r-- 1       13 Oct 30 16:38 VERSION
lrwxrwxrwx 1       33 Nov  1 10:35 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 intermediate format. Ungrib may be run by simply typing the following:

```
>ungrib.exe >& ungrib.log
```

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

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

will be written to the end of the ungrib.log 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`*.

```
>ls

drwxr-xr-x 2       512 Oct 30 16:38 arch
-rwxr-xr-x 1      1672 Sep  8 18:50 clean
-rwxr-xr-x 1      3349 Sep 12 11:11 compile
-rw-r--r-- 1    100168 Nov  1 10:02 compile.output
-rwxr-xr-x 1      4257 Jul 19 13:47 configure
-rw-r--r-- 1      2465 Nov  1 10:00 configure.wps
-rw-r--r-- 1  38869928 Nov  1 10:44 FILE:2006-08-16_12
-rw-r--r-- 1  38869928 Nov  1 10:44 FILE:2006-08-16_18
-rw-r--r-- 1   1831720 Nov  1 10:22 geo_em.d01.nc
-rw-r--r-- 1   4443720 Nov  1 10:22 geo_em.d02.nc
drwxr-xr-x 5       512 Nov  1 10:02 geogrid
lrwxrwxrwx 1        23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
-rw-r--r-- 1      9672 Nov  1 10:22 geogrid.log
lrwxrwxrwx 1        39 Nov  1 10:35 GRIBFILE.AAA ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_00
lrwxrwxrwx 1        39 Nov  1 10:35 GRIBFILE.AAB ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_06
-rwxr-xr-x 1      1138 Aug  3 10:09 link_grib.csh
drwxr-xr-x 4       512 Nov  1 10:02 metgrid
lrwxrwxrwx 1        23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
-rw-r--r-- 1      1638 Nov  1 10:42 namelist.wps
-rw-r--r-- 1      5074 Sep 15 13:05 README
drwxr-xr-x 7       512 Oct 30 16:38 test_suite
drwxr-xr-x 4       512 Nov  1 10:02 ungrib
lrwxrwxrwx 1        21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
-rw-r--r-- 1     29754 Nov  1 10:44 ungrib.log
drwxr-xr-x 3       512 Nov  1 10:02 util
-rw-r--r-- 1        13 Oct 30 16:38 VERSION
lrwxrwxrwx 1        33 Nov  1 10:35 Vtable ->
ungrib/Variable_Tables/Vtable.GFS
```

Step 3: Horizontally interpolating meterological 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 = '2006-08-16_12:00:00','2006-08-16_12:00:00',
 end_date   = '2006-08-16_18:00:00','2006-08-16_12:00:00',
 interval_seconds = 21600
 io_form_geogrid = 2,
 opt_output_from_geogrid_path = './',
 debug_level = 0
/

&metgrid
 fg_name                         = './FILE'
 io_form_metgrid                 = 2,
 opt_output_from_metgrid_path = './',
 opt_metgrid_tbl_path         = 'metgrid/',
/
```

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 not, the WRF dynamical core, number of domains, starting and ending times, and path to the domain files must be set in the "share" namelist record.

In the "metgrid" namelist record, the path and prefix of the intermediate meteorological data files must be given with `fg_name`, and the output format for the horizontally interpolated files should 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.

After suitably editing the namelist.wps file, 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 not set to './'). These files will be named `met_em.d0N.YYYY-MM-DD_HH:00:00` in the case of ARW domains, where `N` is the number of the nest whose data reside in the file, or

met_nmm.d0N.YYYY-MM-DD_HH:00:00 in the case of NMM domains. Here, YYYY-MM-DD_HH:00:00 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       512 Oct 30 16:38 arch
      -rwxr-xr-x 1      1672 Sep  8 18:50 clean
      -rwxr-xr-x 1      3349 Sep 12 11:11 compile
      -rw-r--r-- 1    100168 Nov  1 10:02 compile.output
      -rwxr-xr-x 1      4257 Jul 19 13:47 configure
      -rw-r--r-- 1      2465 Nov  1 10:00 configure.wps
      -rw-r--r-- 1  38869928 Nov  1 10:44 FILE:2006-08-16_12
      -rw-r--r-- 1  38869928 Nov  1 10:44 FILE:2006-08-16_18
      -rw-r--r-- 1   1831720 Nov  1 10:22 geo_em.d01.nc
      -rw-r--r-- 1   4443720 Nov  1 10:22 geo_em.d02.nc
      drwxr-xr-x 5       512 Nov  1 10:02 geogrid
      lrwxrwxrwx 1        23 Nov  1 10:02 geogrid.exe ->
geogrid/src/geogrid.exe
      -rw-r--r-- 1      9672 Nov  1 10:22 geogrid.log
      lrwxrwxrwx 1        39 Nov  1 10:35 GRIBFILE.AAA ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_00
      lrwxrwxrwx 1        39 Nov  1 10:35 GRIBFILE.AAB ->
/mmmtmp/mm5rt/data/gfs/gfs_060816_12_06
      -rwxr-xr-x 1      1138 Aug  3 10:09 link_grib.csh
      -rw-r--r-- 1   5092340 Nov  1 10:58 met_em.d01.2006-08-
16_12:00:00.nc
      -rw-r--r-- 1   5092340 Nov  1 10:58 met_em.d01.2006-08-
16_18:00:00.nc
      -rw-r--r-- 1  12356572 Nov  1 10:58 met_em.d02.2006-08-
16_12:00:00.nc
      drwxr-xr-x 4       512 Nov  1 10:02 metgrid
      lrwxrwxrwx 1        23 Nov  1 10:02 metgrid.exe ->
metgrid/src/metgrid.exe
      -rw-r--r-- 1     62170 Nov  1 10:58 metgrid.log
      -rw-r--r-- 1      1638 Nov  1 10:58 namelist.wps
      -rw-r--r-- 1      5074 Sep 15 13:05 README
      drwxr-xr-x 7       512 Oct 30 16:38 test_suite
      drwxr-xr-x 4       512 Nov  1 10:02 ungrib
      lrwxrwxrwx 1        21 Nov  1 10:02 ungrib.exe ->
ungrib/src/ungrib.exe
      -rw-r--r-- 1     29754 Nov  1 10:44 ungrib.log
      drwxr-xr-x 3       512 Nov  1 10:02 util
      -rw-r--r-- 1        13 Oct 30 16:38 VERSION
      lrwxrwxrwx 1        33 Nov  1 10:35 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 the sole 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.

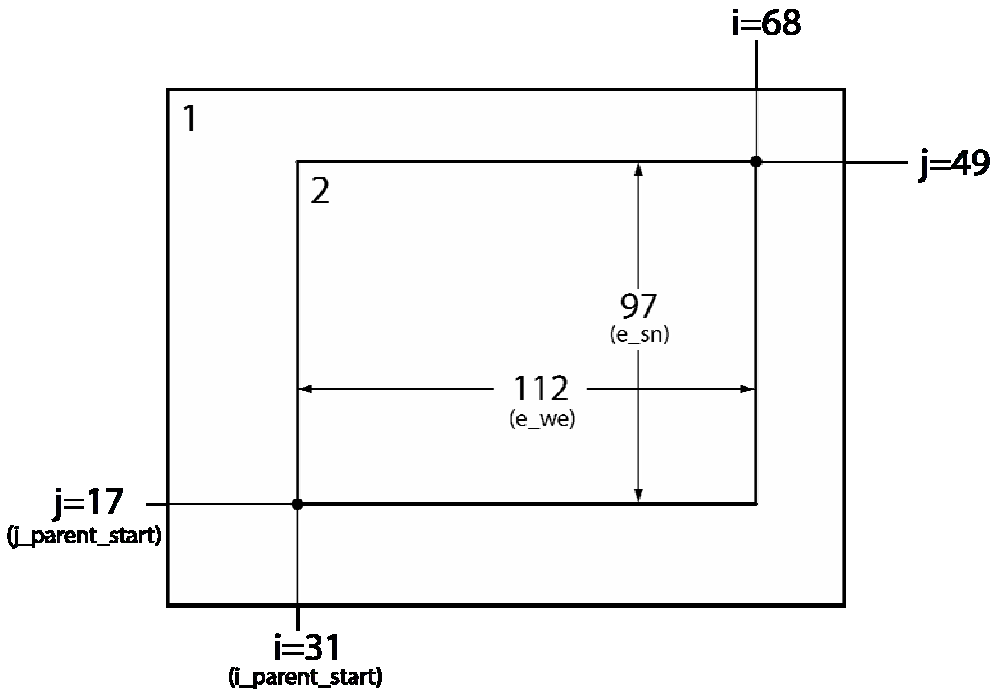
```
&share
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2006-08-16_12:00:00','2006-08-16_12:00:00',
 end_date   = '2006-08-16_18:00:00','2006-08-16_12:00:00',
 interval_seconds = 21600
 io_form_geogrid = 2,
 opt_output_from_geogrid_path = './',
 debug_level = 0
/

&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 = '/data3a/mp/gill/DATA/GEOG'
 opt_geogrid_tbl_path = 'geogrid/'
/
```

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), though the effect on the namelist variables generalize to multiple nests in the obvious way; that is, 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, say *N*, all of the other affected namelist variables must be given a list of *N* values, one for each nest. 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. 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 parent of a nest is the nest refinement ratio with respect to a nest's 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 is shown in the figure above, where it may be seen how each of the above-mentioned variables is found. 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 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 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 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:

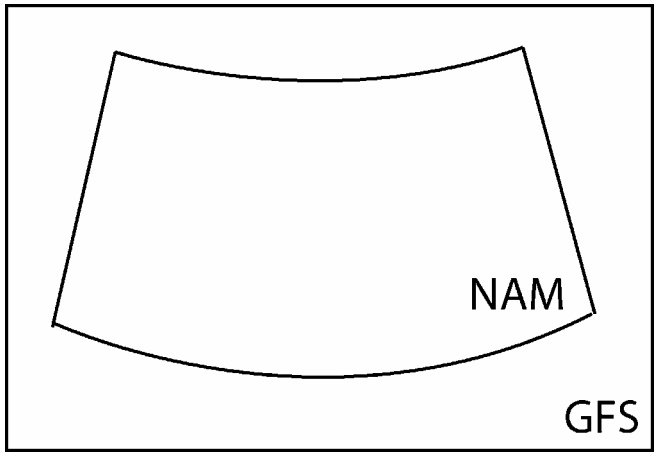
```
&metgrid
 constants_name = 'NARR_FIXED',
 fg_name = 'NARR_3D', 'NARR_SFC'
/
```

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 command 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.exe 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.exe, 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.exe 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 WPS, and are compiled in the `util` directory. These utilities may be used to examine data files, visualize the location of nested domains, 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. 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.

D. 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.

E. 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.

F. 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.

G. 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. 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
                              !     5 = Polar stereographic
integer :: 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 ECWRF 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.exe program and real.exe, 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*.

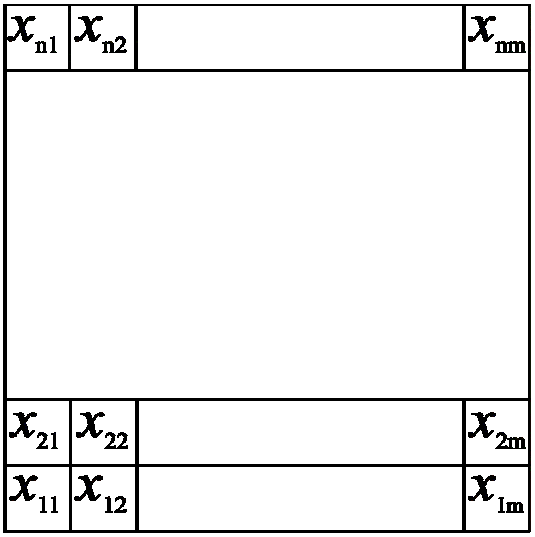
The final group of fields, which provide GRIB2-specific information, are found under the column headings below.

GRIB2 Discp	GRIB2 Catgy	GRIB2 Param	GRIB2 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 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 (smallest index) 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). 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 may be done by first scaling all element 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 positive elements are unaffected by this second conversion. 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 continguously 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 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 as 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 $\times$ 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 42001-43200.42001-43200.

Clearly, since the starting and ending indices 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.

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 second part of the section on GEOGRID.TBL keywords.

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 given 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 given 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 threshold for sending debugging information to standard output. The useful range for the debugging level is 0 to 1000, with higher values permitting more output to be sent to standard output. 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, `parent_id` 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 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 integer *n*). No default value.

9. GEOG_DATA_RES : A list of MAX_DOM character strings specifying, for each nest, a corresponding resolution of source data to be used when interpolating static terrestrial data to the nest's grid. For each nest, this string should match 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 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. 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 NMM, the grid distance is in degrees longitude. Grid distances for nests are determined recursively based on values specified for `parent_grid_ratio`. Default value is 10000.

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 NMM, the grid distance is in degrees longitude. Grid distances for nests are determined recursively based on values specified for parent_grid_ratio. Default value is 10000.

12. MAP_PROJ : A character string specifying the projection of the simulation domain. For ARW, accepted projections are 'Lambert', 'polar', and 'Mercator'; 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. When `wrf_core` = 'ARW', west longitudes are negative, and when `wrf_core` = 'NMM', west longitudes are positive; 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 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. 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.

121. 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*. 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. 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. 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 signs (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. 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`. 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. 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. 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.

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', 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', 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', 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'.

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 signs (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. 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 'all'. *FIELD* may either be the name of another field, 'const', or '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_MSG 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).

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 4: WRF Initialization

Table of Contents

- Introduction
- Initialization for Ideal Data Cases
- Initialization for Real Data Cases

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 WRF model itself is not altered by choosing one initialization over another, but the WRF pre-processors are specifically built based upon a user's selection.

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

- Ideal cases
  - 3d
    - em_b_wave - barclinic wave
    - em_quarter_ss - super cell
  - 2d
    - em_grav2d_x
    - em_hill2d_x
    - em_squall2d_x
    - em_squall2d_y
- Real data cases
  - em_real

The selection of the type of forecast is made when issuing the `./compile` statement. If the user chooses `./compile em_real`, then the initialization program is built using the target module (one of the `./WRFV2/dyn_em/module_initialize_*.F` files). In each of these modules, the same sort of activities go on:

- read data from the namelist
- allocate space
- compute a base state
- compute the perturbations from the base state
- initialize rest of variables

- generate initial condition file

The real-data case does some additional work:

- read input data from the WRF Preprocessing System (WPS) (or SI)
- if it is WPS data, compute dry surface pressure, model levels, and vertically interpolate data
- compute reference temperature profile (differently than with ideal cases, to allow for seasonal norms)
- with input total dry surface pressure, partition into base and perturbation mu
- prepare soil fields for use in model (usually, vertical interpolation to the requested levels)
- checks 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
- 3d boundary data (u, v, t, q, ph) are coupled with map factors (on the correct staggering) and total mu

Both the real.exe and the 3d ideal.exe programs may be run as a distributed memory jobs.

Initialization for Ideal Cases

The program "ideal" is an alternative in the WRF system to running with real-data inputs (which uses program "real".) Typically this program requires no inputs except for the namelist.input and the input_sounding files (except for the b_wave case which uses a 2d binary sounding file). The program outputs the wrfinput_d01 file that is read by the WRF model executable ("wrf.exe".)

Idealized runs can use any of the boundary conditions except "specified", and are not, by default, set up to run with sophisticated physics apart from microphysics. There are no radiation, surface fluxes or frictional effects in these cases, so they are mostly useful for dynamical studies 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, and microphysics options.

The input_sounding 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). The following lines are five digits: height (meters above sea-level), potential temperature (K), mixing ratio (g/kg), x wind component, y wind component (m/s). 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. The base state sounding for idealized cases is the initial sounding minus the

moisture, and so does not have to be defined separately. [b_wave Note: For the b_wave case the input_sounding is not used because the initial 3d arrays are read in from file input_jet. This means for b_wave the namelist.input file cannot be used to change the dimensions either.]

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 module_initialize_[case].F, where [case] is the case chosen in compilation, e.g. module_initialize_squall2d_x.F. The subroutine 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.

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.

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). The data from the WPS originally came 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 through 25 1200
- 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 and input into the real.exe program. The "d01" part of the name is used to identify to which domain this data refers. The trailing characters are the date, where each WPS output file has only a single time-slice of processed data.

The WPS package delivers data that is ready to be used in the WRF system.

- The data adheres to the WRF IO API.
- The data has already been horizontally interpolated to the correct grid-point staggering for each variable.
- 3D meteorological data from the WPS: u, v, theta, mixing ratio
- 3D surface data from the WPS: soil temperature, soil moisture, soil liquid
- 2D meteorological data from the WPS: total dry surface pressure minus ptop
- 2D static data: LOTS! terrain, land categories, soil info, map factors, Coriolis, projection rotation, temporally interpolated monthly data
- 1D array of the vertical coordinate
- constants: domain size, date, lists of available optional fields, corner lat/lons

See next chapter for compile and run WRF initialization program.

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 (regenerated for V2.0 WRF)" 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, so that ./WRFV2/main/real.exe and ./WRFV2/main/wrf.exe exist
- in the ./WRFV2/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 ./WRFV2/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, this should take a few minutes (only a 12 h forecast is requested in the namelist)
- the output file wrfout_d01:2000-01-24_12:00:00 should contain a 12 h forecast at 3 h intervals

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 5: WRF Model

Table of Contents

- Introduction
- Software Requirement
- Before You Start
- How to Compile WRF?
- How to Run WRF?
  - Idealized Case
  - Real Data Case
  - Restart Run
  - Two-Way Nest
  - One-Way Nest
  - Moving Nest
  - Three-dimensional Analysis Nudging
  - Observation Nudging
- Check Output
- Physics and Diffusion Options
- Description of Namelist Variables
- List of Fields in WRF Output

Introduction

The WRF model is a fully compressible, nonhydrostatic model (with a 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 directions. 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 2.2 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
- Non-hydrostatic and hydrostatic (runtime option)
- One-way, two-way nesting and moving nest
- Three-dimensional analysis nudging
- Observation nudging
- Applications ranging from meters to thousands of kilometers

Software requirement

- Fortran 90 or 95 and c compiler
- perl 5.04 or better
- If MPI and OpenMP compilation is desired, it requires MPI or OpenMP libraries
- WRF I/O API supports netCDF, PHD5 and GriB 1/2 formats, hence one of these libraries needs to be available on the computer where you compile and run WRF

Before you start

Before you compile WRF code on your computer, check to see if you have netCDF library installed. This is because one of the supported WRF I/O is netCDF format. If your netCDF is installed in some odd places (e.g. not in your /usr/local/), then you need to know the paths to netCDF library, and to its include/ directory. You may use the environment variable NETCDF to define where the path to netCDF library is. To do so, type

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

If you don't have netCDF on your computer, you need to install it first. You may download netCDF source code or pre-built binary. Installation instruction can be found on the Unidata Web page at http://www.unidata.ucar.edu/.

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

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

***Hint*: NCAR IBM users:**

On one of the NCAR's IBM computer (*bluesky*), the netCDF library is installed for both 32-bit and 64-bit memory usage. The default would be the 32-bit version. If you would like to use the 64-bit version, set the following environment variable before you start compilation:

```
setenv OBJECT_MODE 64
```

This will create correct links to the netCDF library and include file in *netcdf_links*/ directory under *WRFV2*/.

On the other NCAR's IBM, *bluevista*, the netCDF is located in the 'usual' location, /usr/local, and only the 64-bit libraries are installed. No need to set anything prior to compilation.

How to compile WRF?

WRF source code tar file may be downloaded from http://www.mmm.ucar.edu/wrf/download/get_source.html. Once you obtain the tar file, gunzip, and untar the file, and this will create a WRFV2/ directory. This contains:

`Makefile`	Top-level makefile
`README`	General information about WRF code
`README_test_cases`	Explanation of the test cases
`Registry/`	Directory for WRF Registry file
`arch/`	Directory where compile options are gathered
`clean`	script to clean created files, executables
`compile`	script for compiling WRF code
`configure`	script to configure the *configure.wrf* file for compile
`chem/`	Developer code, not yet supported
`dyn_em/`	Directory for modules for dynamics in current WRF core (Advanced Research WRF core)
`dyn_exp/`	Directory for a 'toy' dynamic core
`dyn_nmm/`	NCEP NMM core, 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 7 test case directories,

	may be used to run WRF
`tools/`	Directory that contains tools for

Go to WRFV2 (top) directory.

Type

```
configure, or ./configure
```

and you will be given a list of choices for your computer. These choices range from compiling for a single processor job, to using OpenMP shared-memory or distributed-memory parallelization options for multiple processors. Some options support nesting, others do not. Some use RSL-LITE, others use RSL. So select the option carefully. For example, the choices for a Linux computer looks like this:

```
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.

------------------------------------------------------------------------

Please select from among the following supported platforms.

1. PC Linux i486 i586 i686, PGI compiler (Single-threaded, no nesting)
2. PC Linux i486 i586 i686, PGI compiler (single threaded, allows nesting using
RSL without MPI)
3. PC Linux i486 i586 i686, PGI compiler SM-Parallel (OpenMP, no nesting)
4. PC Linux i486 i586 i686, PGI compiler SM-Parallel (OpenMP, allows nesting
using RSL without MPI)
5. PC Linux i486 i586 i686, PGI compiler DM-Parallel (RSL, MPICH, Allows
nesting)
6. PC Linux i486 i586 i686, PGI compiler DM-Parallel (RSL_LITE, MPICH, Allows
nesting)
7. Intel xeon i686 ia32 Xeon Linux, ifort compiler (single-threaded, no
nesting)
8. Intel xeon i686 ia32 Xeon Linux, ifort compiler (single threaded, allows
nesting using RSL without MPI)
9. Intel xeon i686 ia32 Xeon Linux, ifort compiler (OpenMP)
10. Intel xeon i686 ia32 Xeon Linux, ifort compiler SM-Parallel (OpenMP, allows
nesting using RSL without MPI)
11. Intel xeon i686 ia32 Xeon Linux, ifort+icc compiler DM-Parallel (RSL,
MPICH, allows nesting)
12. Intel xeon i686 ia32 Xeon Linux, ifort+gcc compiler DM-Parallel (RSL,
MPICH, allows nesting)
13. PC Linux i486 i586 i686, PGI compiler, ESMF (Single-threaded, ESMF
coupling, no nesting)

Enter selection [1-13] :
```

Enter a number for an option that is best for your computer and application.

You will see a **configure.wrf** file created. Edit compile options/paths, if necessary.

Hint: **for choosing RSL_LITE versus RSL:**

Choose compile options that use RSL_LITE whenever one can. The only option RSL_LITE doesn't support is the periodic boundary condition in y. RSL_LITE is a bit faster, and will work for domains dimensioned greater than 1024x1024. The positive-definite advection scheme is implemented using RSL_LITE only.

Hint: **for nesting compile:**

- On most platforms, this requires RSL_LITE or RSL/MPI, even if you only have one processor. Check the options carefully and select those which support nesting.

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

```
setenv WRF_EM_CORE 1
```

Type '**compile**', or '`./compile`' and it will show the choices:

```
  Usage:

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

          test cases (see README_test_cases for details):

           compile em_b_wave
           compile em_grav2d_x
           compile em_hill2d_x
           compile em_quarter_ss
           compile em_real
           compile em_squall2d_x
           compile em_squall2d_y
          compile -h                help message
```

where **em** stands for the Advanced Research WRF dynamic solver (which currently is the 'Eulerian mass-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 for the initialization programs (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 clean directories of all object files and executables, type '***clean***'.

Type '***clean -a***' to remove all built files, 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

Type '***compile case_name***' to compile. Suppose you would like to run the 2-dimensional squall case, type

```
compile em_squall2d_x
```

or

```
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/** and **run/** directories. cd to either directory to run the model.

b. Real-data case

Compile WRF model after '***configure***', type

```
compile em_real
```

or

```
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 directory to run the model.

How to run WRF?

After a successful compilation, it is time to run the model. You can do so by either cd to the **run/** directory, or the **test/**case_name directory. In either case, you should see executables, **ideal.exe** or **real.exe**, and **wrf.exe**, linked files (mostly for real-data cases), and one or more **namelist.input** files in the directory.

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 you choose to compile the test case **em_squall2d_x**, now type `'cd test/em_squall2d_x'` or `'cd run'` to go to a working directory.

Edit **namelist.input** file (see README.namelist in WRFV2/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 data files that you might need to run the case.

To run the initialization program, type

```
ideal.exe
```

This will generate **wrfinput_d01** file in the same directory. All idealized cases do not require lateral boundary file because of the boundary condition choices they use, such as the periodic boundary condition option.

Note:

- **ideal.exe** cannot generally be run in parallel. For parallel compiles, run this on a single processor.
- The exception is the **quarter_ss** case, which can now be run in MPI.

To run the model, type

```
wrf.exe
```

or variations such as

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

Note:

- Two-dimensional ideal cases cannot be run in MPI parallel. OpenMP is ok.

- The execution command may be different for MPI runs on different machines, e.g. mpirun.

After successful completion, you should see ***wrfout_d01_0001-01-01**** and **wrfrst*** files, depending on how one specifies the namelist variables for output.

b. Real-data case

Type '`cd test/em_real`' or '`cd run`', and this is where you are going to run both the WRF initialization program, **real.exe**, and WRF model, **wrf.exe**. Start with a namelist.input file template in the directory, edit it to fit your case.

- Running real.exe using WPS output

Running a real-data case requires successfully running the **WRF Preprocessing System** programs (or WPS) and make sure **met_em.*** files from WPS are seen in the run directory. 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 for you.

Other options for use to assist vertical interpolation are:

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.

- Running real.exe using SI output

Running a real-data case requires successfully running the WRF **Standard Initialization** program. Make sure **wrf_real_input_em.*** files from the Standard Initialization are in this directory (you may link the files to this directory).

If you use SI, you must use the SI version 2.0 and above, to prepare input for V2 WRF! Make sure you also have this line in your namelist.input file (the default input file is expected to come from WPS):

auxinput_inname = "wrf_real_input_em.d<domain>.<date>"

For both WPS and SI output, edit **namelist.input** for start and end dates, and domain dimensions. Also edit time step, output, nest and physics options.

Type '**real.exe**' to 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

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

For DM (distributed memory) parallel systems, some form of **mpirun** command will be needed here. 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 (such NCAR's IBM *bluesky*) the command may be:

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

in 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 *bluevista*)

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 minutes) to be equal to or less than the simulation length, 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 the restart time (and the time a restart file is written). The other namelist variable one must set is *restart*, this variable should be set to .true. for a restart run.

d. Two-way Nested Run

WRF V2 supports a two-way nest option in both 3-D idealized cases (quarter_ss and b_wave) and real data cases. The model can handle multiple domains at the same nest level (no overlapping nest), and multiple nest levels (telescoping). A moving nest option has also been available since V2.0.3.1.

Make sure that you compile the code with nest options.

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***. The following are the key namelist variables to modify:

start_ and ***end_year/month/day/minute/second***: these control the nest start and end times

input_from_file: whether a nest requires an input file (e.g. *wrfinput_d02*). This is typically an option for a real data case.

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 temp, soil moisture and temperature).

max_dom: setting this to a number > 1 will invoke nesting. For example, if you want to have one coarse domain and one nest, set this variable to 2.

grid_id: domain identifier will be used in the wrfout naming convention.

parent_id: use the grid_id to define the parent_id number for a nest. The most coarse grid must be *grid_id* = 1.

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 in namelist.wps, if you use WPS. (If you use SI, you should find these numbers in your SI's $MOAD_DATAROOT/static/wrfsi.nl namelist file, and look for values in the second (and third, and so on) column of DOMAIN_ORIGIN_LLI and DOMAIN_ORIGIN_LLJ).

parent_grid_ratio: integer parent-to-nest domain grid size ratio. If *feedback* is off, then this ratio can be even or odd. If *feedback* is on, then this ratio has to be odd.

parent_time_step_ratio: time ratio for the coarse and nest domains may be different from the ***parent_grid_ratio***. For example, you may run a coarse domain at 30 km, and a nest at 10 km, the parent_grid_ratio in this case is 3. But you do not have to use 180 sec for the coarse domain and 60 for the nest domain. You may use, for example, 45 sec or 90 sec for the nest domain by setting this variable to 4 or 2.

feedback: this option takes the values of prognostic variables (average of cell for mass points, and average of the cell face for horizontal momentum points) in the nest and overwrites the values in the coarse domain at the coincident points. This is the reason currently that it requires odd parent_grid_ratio with this option.

smooth_option: this a smoothing option for the parent domain if ***feedback*** is on. Three options are available: 0 = no smoothing; 1 = 1-2-1 smoothing; 2 = smoothing-desmoothing. (There was a bug for this option in pre-V2.1 code, and it is fixed.)

3-D Idealized Cases

For 3-D idealized cases, no additional 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 (namelist variable ***input_from_file*** set to F for each domain).

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 nested wrfinput file to read in (similar to MM5 nest option IOVERW = 1). The limitation of this option is that this only allows the nest to start at hour 0 of the coarse domain run.

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 current Registry setting which designates certain fields to be read in from auxiliary input stream 2. This option allows the nest initialization to use 3-D meterological fields interpolated from the coarse domain, static fields and masked, time-varying surface fields from 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. This option was introduced in V2.1.

To run `real.exe` for a nested run, one must first run WPS (or SI) and create data for all the nests. Suppose one has run WPS for a two-domain nest case, and created these files 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
```

Link or move all these files to the run directory, which could be `run/` or `em_real/`.

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

```
real.exe
```

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
```

The way to create nested input files has been greatly simplified due to some improvement introduced to WRF Version 2.1.2 (released in January 2006).

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

WRF supports one-way nested option. One-way nesting is defined as a finer grid resolution run made as a subsequent run after the coarser grid resolution run and driven by coarse grid output as initial and lateral boundary conditions, together with input from higher resolution terrestrial fields (e.g. terrain, landuse, etc.), and masked surface fields (such as soil temperature and moisture).

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

It takes several steps to make a one-way nested run. It involves these steps:

1) Make a coarse grid run
2) Make temporary fine grid initial condition (only a single time period is required)
3) Run program ***ndown,*** with coarse grid WRF model output, and fine grid input to generate fine grid initial and boundary conditions
4) Make the fine grid run

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, one would have run WPS (or SI) and requested one coarse and one nest domain, and for the one time period one wants to start the one-way nested run. This should generate a WPS output for the nested domain (domain 2) named `met_em.d01.*` (or `wrf_real_input_em.d02.*` from SI).

- Rename `met_em.d02.*` to `met.d01.*`. (Move the original domain 1 WPS output files to a different place before you do this.)
- Edit the *namelist.input* file for this domain (pay attention to column 1 only,) and edit in the correct start time, grid dimensions and physics options.
- Run **real.exe** for this domain and this will produce a `wrfinput_d01` file.
- Rename this `wrfinput_d01` file to `wrfndi.d02`. Note that this is a new change in V2.2.

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 frequencey (namelist variable interval_seconds) is the time in seconds between the coarse grid output times.
- Run **ndown.exe**, with inputs from the coarse grid `wrfout` files, and `wrfndi.d02` file generated from Step 2 above. This will produce `wrfinput_d02` and `wrfbdy_d02` files.

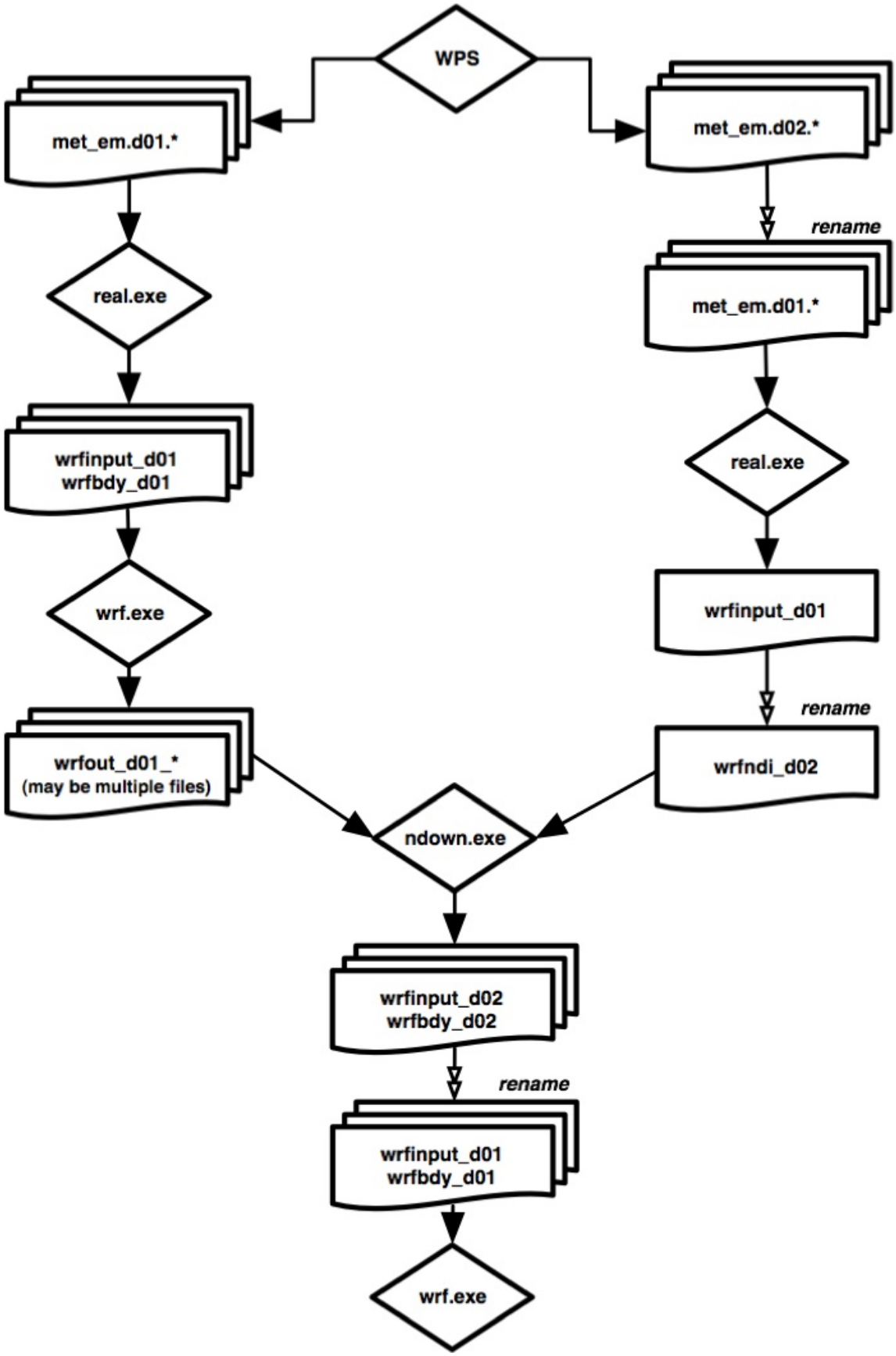
Note that one may run program ndown in mpi - if it is compiled so. For example,

```
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 only.
- Run WRF for this grid.

The following figure summarizes the data flow for a one-way nested run.


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

f. Moving-Nest Run

The moving nest option is supported in the current WRF. Two types of moving tests are allowed. 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 runs, one first needs to compile the code with -DMOVE_NESTS flag added to ARCHFLAGS in the *configure.wrf* file. 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).

To make the automatic moving nest runs, two compiler flags are needed in ARCHFLAGS: -DMOVE_NESTS and -DVORTEX_CENTER. (***Note*** that this compile would only support auto-moving nest runs, 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).

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

This option is introduced in V2.2.

Prepare input data to WRF as usual whether you use WPS or SI. If you would like to nudge in the nest domains as well, make sure you process all time periods for all domains.

Before you run real.exe, set the following options, 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_d01 and wrfbdy_d01 files, a file named "wrffdda_d01" by default. 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 WRFV2/test/em_real/ for more information.

h. Observation Nudging Run

This option is introduced in V2.2.

In addition to the usual input data preparation from running WPS or SI, one needs to prepare an observation files to be used in the model. See http://www.mmm.ucar.edu/wrf/users/wrfv2/How_to_run_obs_fdda.html for instructions.

Once one has the observation file(s) prepared, one can activate the observation nudging options in WRF:

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

See http://www.mmm.ucar.edu/wrf/users/wrfv2/How_to_run_obs_fdda.html and README.obs_fdda in WRFV2/test/em_real/ for more information.

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 run successfully.
- 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 you may have created. This file logs the time taken to compute for one model time step, and to write one history 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 violate the CFL criterion. 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.

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 2.2.

1. Microphysics *(mp_physics)*

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

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

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

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

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

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

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

h. NCEP 3-class: An older version of (c)

i. NCEP 5-class: An older version of (d)

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.

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

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

2.2 Shortwave Radiation *(ra_sw_physics)*

a. Dudhia scheme: Simple downward integration allowing efficiently for clouds and clear-sky absorption and scattering.

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

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

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

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.

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

3.2 Land Surface *(sf_surface_physics)*

a. 5-layer thermal diffusion: Soil temperature only scheme, using five layers.

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.

- 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.

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.

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

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

5. Cumulus Parameterization *(cu_physics)*

a. Kain-Fritsch scheme: Deep and shallow sub-grid scheme using a mass flux approach with downdrafts and CAPE removal time scale.

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

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

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.

b. Full diffusion: Gradients use full metric terms to more accurately compute horizontal gradients in sloped coordinates.

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.

b. 3d TKE: A prognostic equation for turbulent kinetic energy is used, and K is based on TKE.

c. 3d Deformation: K is diagnosed from 3d deformation and stability following a Smagorinsky approach.

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.

1.3 6^{th} Order Horizontal Diffusion *(diff_6th_opt)*

6^{th}-order horizontal hyperdiffusion (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 or a Rayleigh relaxation layer 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.

3. Advection Options

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

b. Vertical advection orders for momentum (v_mom_adv_order) and scalar (v_sca_adv_order) can be 2^{nd} and 6th, with 3^{rd} 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).

4. 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).
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.).

Description of Namelist Variables

The following is a description of namelist variables. The variables that are function of nest are indicated by *(max_dom)* following the variable. Also see README.namelist in WRFV2/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
Auxinput1_inname	“met_em.d<domain>_<date>”	input from WPS (this is the default)
	“wrf_real_input_em.d<domain>_<date>”	input from SI
io_form_history	2	2 = netCDF; 102 = split netCDF files one per processor (no supported post-processing software for split files)
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
	4	PHDF5 format (no supported post-processing software)
	5	GRIB1 format (no supported post-processing software)

	1	binary format (no supported post-processing software)
debug_level	0	50,100,200,300 values give increasing prints
auxhist2_outname	"rainfall"	file name for extra output; if not specified, auxhist2_d_ 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
auxinput11_interval		
auxinput11_end_h		
nocolons	.false.	replace : with _ in output file names
write_input	t	write input-formatted data as output for 3DVAR apllication
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 date
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 varialbes are on unstaggered levels. Vertical dimensions need to be the same for all nests.
num_metgrid_levels	40	number of vertical levels in the incoming data: type nudump –h to find out (WPS data only)
eta_levels	1.0, 0.99,…0.0	model eta levels (WPS data only). If a user does not specify this, 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
interp_type	1	vertical interpolation; 1: linear in pressure; 2: linear in log(pressure)
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.	used in mass model 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)	0	starting LLC I-indices from the parent domain
j_parent_start (max_dom)	0	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
		Namelist variables for controling the prototype moving nest: Note that moving nest needs to be activated at the compile time by adding -DMOVE_NESTS to the ARCHFLAGS. The maximum number of moves, max_moves, is set to be 50, but can be modified in source code file *frame/module_driver_constants.F*
num_moves	2,	total number of moves
move_id	2,2,	a list of nest domain id's, one per move
move_interval	60,120,	time in minutes since the start of this domain
move_cd_x	1,-1,	the number of parent domain grid cells to move in i direction
move_cd_y	-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. The limitation now is to move only 1 grid cell at each move.

vortex_interval	15	how often the new vortex position is computed
max_vortex_speed	40	used to compute the search radius for the new vortex position
corral_dist	8	how many coarse grid cells the moving nest is allowed to get near the coarse grid boundary
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
	8	new Thompson graupel scheme
	98	NCEP 3-class simple ice scheme (to be removed)
	99	NCEP 5-class scheme (to be removed)
mp_zero_out		For non-zero mp_physics options, to keep Qv >= 0, and to set the other moisture fields < a threshold value to zero
	0	no action taken, no adjustment to any moist field
	1	except for Qv, all other moist arrays are set to zero if they fall below a critical value

	2	Qv is >= 0, all other moist 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 moist arrays (except for Qv) are set to zero (unit: kg/kg)
`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)
`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 clearsky 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
`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	Noah land-surface model

	3	RUC land-surface model
`bl_pbl_physics (max_dom)`		boundary-layer option
	0	no boundary-layer
	1	YSU scheme
	2	Mellor-Yamada-Janjic (Eta) TKE scheme
	99	MRF scheme (to be removed)
`bldt (max_dom)`	0	minutes between boundary-layer physics calls
`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
	99	previous Kain-Fritsch scheme
`cudt`	0	minutes between cumulus physics calls
`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 = SI/gridgen, 2 = GRIB data from another model (only possible if VEGCAT/SOILCAT are in wrf_real_input_em files from SI; used in *real*)
`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

ucmcall	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 during a model simulation (set in *real*)
	0	no SST update
	1	*real.exe* will create wrflowinput_d01 file at the same time interval as the available input data. To use it in wrf.exe, add auxinput5_inname = "wrflowinp_d01", auxinput5_interval, and auxinput5_end_h in namelist section *&time_control*
&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)		
if_no_pbl_nudging_uv (max_dom)	0	0= no nudging of u and v in the pbl; 1= nudging in the pbl
if_no_pbl_nudging_t (max_dom)	0	0= no nudging of temp in the pbl; 1= nudging in the pbl

if_no_pbl_nudging_t (max_dom)	0	0= no nudging of qvapor in the pbl; 1= 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.	horizonal radius of influence in km
obs_rinsig	0.1	vertical radius of influence in eta
obs_twindo	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	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_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)
&dynamics		Diffusion, damping options, advection options
dyn_opt	2	dynamical core option: advanced research WRF core (Eulerian mass)
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	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	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 (may now be used in real-data runs)
	0	without damping
	1	with diffusive damping (dampcoef nondimensional ~ 0.01 - 0.1)
	2	with Rayleigh damping (dampcoef inverse time scale [1/s], e.g. 0.003)
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_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)
mix_full_fields	.false.	For diff_opt=2 only, vertical diffusion acts on full fields (not just on perturbation from 1D base profile) (idealized)
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	.false.	positive define advection of moisture; set to .true. to turn it on
pd_scalar	.false.	positive define advection of scalars
pd_tke	.false.	positive define advection of tke
pd_chem	.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
&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 applies to domain 1)
		The above 4 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.	nested boundary conditions (inactive)
&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, don't change
&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

List of Fields in WRF Output

List of Fields

The following is an edited output from netCDF command 'ncdump':

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

```
char Times(Time, DateStrLen) ;
float LU_INDEX(Time, south_north, west_east) ;
        LU_INDEX:description = "LAND USE CATEGORY" ;
        LU_INDEX:units = "" ;
float U(Time, bottom_top, south_north, west_east_stag) ;
        U:description = "x-wind component" ;
        U:units = "m s-1" ;
float V(Time, bottom_top, south_north_stag, west_east) ;
        V:description = "y-wind component" ;
        V:units = "m s-1" ;
float W(Time, bottom_top_stag, south_north, west_east) ;
        W:description = "z-wind component" ;
        W:units = "m s-1" ;
float PH(Time, bottom_top_stag, south_north, west_east) ;
        PH:description = "perturbation geopotential" ;
        PH:units = "m2 s-2" ;
float PHB(Time, bottom_top_stag, south_north, west_east) ;
        PHB:description = "base-state geopotential" ;
        PHB:units = "m2 s-2" ;
float T(Time, bottom_top, south_north, west_east) ;
        T:description = "perturbation potential temperature (theta-t0)" ;
        T:units = "K" ;
float MU(Time, south_north, west_east) ;
        MU:description = "perturbation dry air mass in column" ;
        MU:units = "Pa" ;
float MUB(Time, south_north, west_east) ;
        MUB:description = "base state dry air mass in column" ;
        MUB:units = "Pa" ;
float P(Time, bottom_top, south_north, west_east) ;
        P:description = "perturbation pressure" ;
        P:units = "Pa" ;
float PB(Time, bottom_top, south_north, west_east) ;
        PB:description = "BASE STATE PRESSURE" ;
        PB:units = "Pa" ;
float FNM(Time, bottom_top) ;
        FNM:description = "upper weight for vertical stretching" ;
        FNM:units = "" ;
float FNP(Time, bottom_top) ;
        FNP:description = "lower weight for vertical stretching" ;
        FNP:units = "" ;
float RDNW(Time, bottom_top) ;
        RDNW:description = "inverse dn values on full (w) levels" ;
        RDNW:units = "" ;
float RDN(Time, bottom_top) ;
        RDN:description = "dn values on half (mass) levels" ;
        RDN:units = "" ;
float DNW(Time, bottom_top) ;
        DNW:description = "dn values on full (w) levels" ;
        DNW:units = "" ;
float DN(Time, bottom_top) ;
        DN:description = "dn values on half (mass) levels" ;
        DN:units = "" ;
float ZNU(Time, bottom_top) ;
        ZNU:description = "eta values on half (mass) levels" ;
        ZNU:units = "" ;
float ZNW(Time, bottom_top_stag) ;
        ZNW:description = "eta values on full (w) levels" ;
```

```
          ZNW:units = "" ;
float CFN(Time, ext_scalar) ;
          CFN:description = "" ;
          CFN:units = "" ;
float CFN1(Time, ext_scalar) ;
          CFN1:description = "" ;
          CFN1:units = "" ;
float EPSTS(Time, ext_scalar) ;
          EPSTS:description = "" ;
          EPSTS:units = "" ;
float Q2(Time, south_north, west_east) ;
          Q2:description = "QV at 2 M" ;
          Q2:units = "kg kg-1" ;
float T2(Time, south_north, west_east) ;
          T2:description = "TEMP at 2 M" ;
          T2:units = "K" ;
float TH2(Time, south_north, west_east) ;
          TH2:description = "POT TEMP at 2 M" ;
          TH2:units = "K" ;
float PSFC(Time, south_north, west_east) ;
          PSFC:description = "SFC PRESSURE" ;
          PSFC:units = "Pa" ;
float U10(Time, south_north, west_east) ;
          U10:description = "U at 10 M" ;
          U10:units = "m s-1" ;
float V10(Time, south_north, west_east) ;
          V10:description = "V at 10 M" ;
          V10:units = "m s-1" ;
float RDX(Time, ext_scalar) ;
          RDX:description = "INVERSE X GRID LENGTH" ;
          RDX:units = "" ;
float RDY(Time, ext_scalar) ;
          RDY:description = "INVERSE Y GRID LENGTH" ;
          RDY:units = "" ;
float RESM(Time, ext_scalar) ;
          RESM:description = "TIME WEIGHT CONSTANT FOR SMALL STEPS" ;
          RESM:units = "" ;
float ZETATOP(Time, ext_scalar) ;
          ZETATOP:description = "ZETA AT MODEL TOP" ;
          ZETATOP:units = "" ;
float CF1(Time, ext_scalar) ;
          CF1:description = "2nd order extrapolation constant" ;
          CF1:units = "" ;
float CF2(Time, ext_scalar) ;
          CF2:description = "2nd order extrapolation constant" ;
          CF2:units = "" ;
float CF3(Time, ext_scalar) ;
          CF3:description = "2nd order extrapolation constant" ;
          CF3:units = "" ;
int ITIMESTEP(Time, ext_scalar) ;
          ITIMESTEP:description = "" ;
          ITIMESTEP:units = "" ;
float QVAPOR(Time, bottom_top, south_north, west_east) ;
          QVAPOR:description = "Water vapor mixing ratio" ;
          QVAPOR:units = "kg kg-1" ;
float QCLOUD(Time, bottom_top, south_north, west_east) ;
          QCLOUD:description = "Cloud water mixing ratio" ;
          QCLOUD:units = "kg kg-1" ;
float QRAIN(Time, bottom_top, south_north, west_east) ;
          QRAIN:description = "Rain water mixing ratio" ;
          QRAIN:units = "kg kg-1" ;
float LANDMASK(Time, south_north, west_east) ;
          LANDMASK:description = "LAND MASK (1 FOR LAND, 0 FOR WATER)" ;
          LANDMASK:units = "" ;
float TSLB(Time, soil_layers_stag, south_north, west_east) ;
          TSLB:description = "SOIL TEMPERATURE" ;
          TSLB:units = "K" ;
float ZS(Time, soil_layers_stag) ;
          ZS:description = "DEPTHS OF CENTERS OF SOIL LAYERS" ;
          ZS:units = "m" ;
float DZS(Time, soil_layers_stag) ;
```

```
          DZS:description = "THICKNESSES OF SOIL LAYERS" ;
          DZS:units = "m" ;
float SMOIS(Time, soil_layers_stag, south_north, west_east) ;
          SMOIS:description = "SOIL MOISTURE" ;
          SMOIS:units = "m3 m-3" ;
float SH2O(Time, soil_layers_stag, south_north, west_east) ;
          SH2O:description = "SOIL LIQUID WATER" ;
          SH2O:units = "m3 m-3" ;
float XICE(Time, south_north, west_east) ;
          XICE:description = "SEA ICE FLAG" ;
          XICE:units = "" ;
float SFROFF(Time, south_north, west_east) ;
          SFROFF:description = "SURFACE RUNOFF" ;
          SFROFF:units = "mm" ;
float UDROFF(Time, south_north, west_east) ;
          UDROFF:description = "UNDERGROUND RUNOFF" ;
          UDROFF:units = "mm" ;
int IVGTYP(Time, south_north, west_east) ;
          IVGTYP:description = "DOMINANT VEGETATION CATEGORY" ;
          IVGTYP:units = "" ;
int ISLTYP(Time, south_north, west_east) ;
          ISLTYP:description = "DOMINANT SOIL CATEGORY" ;
          ISLTYP:units = "" ;
float VEGFRA(Time, south_north, west_east) ;
          VEGFRA:description = "VEGETATION FRACTION" ;
          VEGFRA:units = "" ;
float GRDFLX(Time, south_north, west_east) ;
          GRDFLX:description = "GROUND HEAT FLUX" ;
          GRDFLX:units = "W m-2" ;
float SNOW(Time, south_north, west_east) ;
          SNOW:description = "SNOW WATER EQUIVALENT" ;
          SNOW:units = "kg m-2" ;
float SNOWH(Time, south_north, west_east) ;
          SNOWH:description = "PHYSICAL SNOW DEPTH" ;
          SNOWH:units = "m" ;
float CANWAT(Time, south_north, west_east) ;
          CANWAT:description = "CANOPY WATER" ;
          CANWAT:units = "kg m-2" ;
float SST(Time, south_north, west_east) ;
          SST:description = "SEA SURFACE TEMPERATURE" ;
          SST:units = "K" ;
float MAPFAC_M(Time, south_north, west_east) ;
          MAPFAC_M:description = "Map scale factor on mass grid" ;
          MAPFAC_M:units = "" ;
float MAPFAC_U(Time, south_north, west_east_stag) ;
          MAPFAC_U:description = "Map scale factor on u-grid" ;
          MAPFAC_U:units = "" ;
float MAPFAC_V(Time, south_north_stag, west_east) ;
          MAPFAC_V:description = "Map scale factor on v-grid" ;
          MAPFAC_V:units = "" ;
float F(Time, south_north, west_east) ;
          F:description = "Coriolis sine latitude term" ;
          F:units = "s-1" ;
float E(Time, south_north, west_east) ;
          E:description = "Coriolis cosine latitude term" ;
          E:units = "s-1" ;
float SINALPHA(Time, south_north, west_east) ;
          SINALPHA:description = "Local sine of map rotation" ;
          SINALPHA:units = "" ;
float COSALPHA(Time, south_north, west_east) ;
          COSALPHA:description = "Local cosine of map rotation" ;
          COSALPHA:units = "" ;
float HGT(Time, south_north, west_east) ;
          HGT:description = "Terrain Height" ;
          HGT:units = "m" ;
float TSK(Time, south_north, west_east) ;
          TSK:description = "SURFACE SKIN TEMPERATURE" ;
          TSK:units = "K" ;
float P_TOP(Time, ext_scalar) ;
          P_TOP:description = "PRESSURE TOP OF THE MODEL" ;
          P_TOP:units = "Pa" ;
```

```
  float RAINC(Time, south_north, west_east) ;
            RAINC:description = "ACCUMULATED TOTAL CUMULUS PRECIPITATION" ;
            RAINC:units = "mm" ;
  float RAINNC(Time, south_north, west_east) ;
            RAINNC:description = "Accumulated Total Grid Scale Precipitation" ;
            RAINNC:units = "mm" ;
  float SWDOWN(Time, south_north, west_east) ;
            SWDOWN:description = "Downward Short Wave Flux At Ground Surface" ;
            SWDOWN:units = "W m-2" ;
  float GLW(Time, south_north, west_east) ;
            GLW:description = "DOWNWARD LONG WAVE FLUX AT GROUND SURFACE" ;
            GLW:units = "W m-2" ;
  float XLAT(Time, south_north, west_east) ;
            XLAT:description = "LATITUDE, SOUTH IS NEGATIVE" ;
            XLAT:units = "degree_north" ;
  float XLONG(Time, south_north, west_east) ;
            XLONG:description = "LONGITUDE, WEST IS NEGATIVE" ;
            XLONG:units = "degree_east" ;
  float TMN(Time, south_north, west_east) ;
            TMN:description = "SOIL TEMPERATURE AT LOWER BOUNDARY" ;
            TMN:units = "K" ;
  float XLAND(Time, south_north, west_east) ;
            XLAND:description = "LAND MASK (1 FOR LAND, 2 FOR WATER)" ;
            XLAND:units = "" ;
  float PBLH(Time, south_north, west_east) ;
            PBLH:description = "PBL HEIGHT" ;
            PBLH:units = "m" ;
  float HFX(Time, south_north, west_east) ;
            HFX:description = "UPWARD HEAT FLUX AT THE SURFACE" ;
            HFX:units = "W m-2" ;
  float QFX(Time, south_north, west_east) ;
            QFX:description = "UPWARD MOISTURE FLUX AT THE SURFACE" ;
            QFX:units = "kg m-2 s-1" ;
  float LH(Time, south_north, west_east) ;
            LH:description = "LATENT HEAT FLUX AT THE SURFACE" ;
            LH:units = "W m-2" ;
  float SNOWC(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`

List of Global Attributes

```
:TITLE = " OUTPUT FROM WRF V2.0.3.1 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 ;
:GRIDTYPE = "C" ;
:DYN_OPT = 2 ;
:DIFF_OPT = 0 ;
:KM_OPT = 1 ;
:DAMP_OPT = 0 ;
:KHDIF = 0.f ;
:KVDIF = 0.f ;
:MP_PHYSICS = 3 ;
:RA_LW_PHYSICS = 1 ;
:RA_SW_PHYSICS = 1 ;
:SF_SFCLAY_PHYSICS = 1 ;
:SF_SURFACE_PHYSICS = 1 ;
:BL_PBL_PHYSICS = 1 ;
:CU_PHYSICS = 1 ;
: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 ;
:DX = 30000.f ;
:DY = 30000.f ;
:DT = 180.f ;
:CEN_LAT = 34.72602f ;
:CEN_LON = -81.22598f ;
:TRUELAT1 = 30.f ;
:TRUELAT2 = 60.f ;
:MOAD_CEN_LAT = 34.72602f ;
:STAND_LON = -98.f ;
:GMT = 12.f ;
:JULYR = 2000 ;
:JULDAY = 24 ;
:MAP_PROJ = 1 ;
:MMINLU = "USGS" ;
:ISWATER = 16 ;
:ISICE = 24 ;
:ISURBAN = 1 ;
:ISOILWATER = 14 ;
```

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 6: WRF-VAR

Table of Contents

- Introduction
- Goals Of This WRF-Var Tutorial
- Tutorial Schedule
- Download Test Data
- The 3D-Var Observation Preprocessor (3DVAR_OBSPROC)
- Setting up WRF-Var
- Run WRF-Var CONUS Case Study
- WRF-Var Diagnostics
- Updating WRF lateral boundary conditions

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 3D-Var web-page http://www.wrf-model.org/development/group/WG4.The documentation you are reading is the "Users Guide" for those interested in downloaded 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;

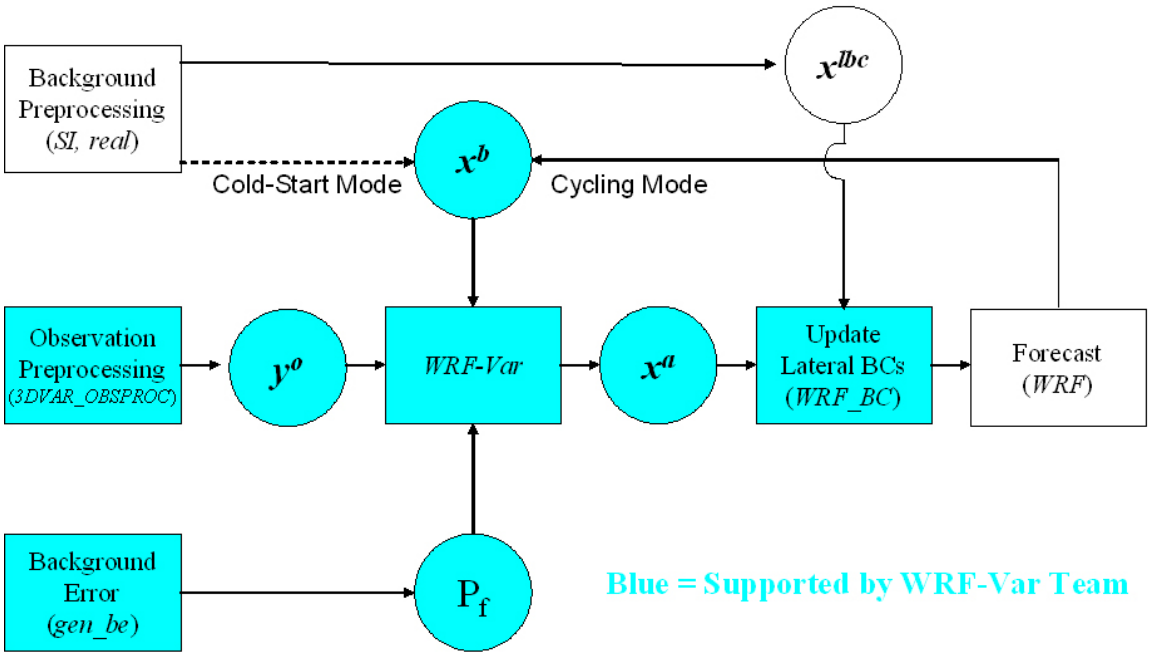
- Users who are looking for references to diagnostics, namelist options etc - look for 'Miscellanies' and 'Trouble Shooting' sections on each page.

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 have 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 about3D-Var, you should first read the WRF-Var tutorial presentations available from the WRF 3D-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 the 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 two input files yourselves.

The 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 so please modify only those that are supported. The ones we support are indicated in the default run script (DA_Run_WRF-Var.csh). Good examples include withholding observations and tuning background error coefficients.
- 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, were quest 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 *Standard Initialization* (SI) 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 datasets to run WRF-Var.
b) The 3D-Var Observation Preprocessor(3DVAR_OBSPROC): Describes how to create an observation file for subsequent use in WRF-Var, and plot observation distributions.
c) Setting upWRF-Var: In this part of the tutorial you will download and compile the codes that form the WRF-Var system (3DVAR_OBSPROC, WRF-Var, WRF_BC).
d) Run WRF-VarCONUS Case Study: In this section, you will learn how to run WRF-Var for a test case.
e) WRF-VarDiagnostics: 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.
f) Updating WRF lateral boundaryconditions: Before using the 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.

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 the SI(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 covariances currently calculated via the *NMC-method*).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 Standard Initialization (SI) Or WRF
Observations	ASCII (BUFR also possible)	Observation Preprocessor (3DVAR_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 the input data sets yourselves.

Downloading Test Data

In the online tutorial, we will store data in a directory defined by the environment variable $*DAT_DIR*. This directory can be in any location that can be read and written to from your current machine and which have at least 100MB of memory available (for this test case, may be more for other cases). Type

setenv DAT_DIR dat_dir

where dat_dir is your chosen directory. Create these directories (if they do not already exist) and type

cd $DAT_DIR

Download the tutorial test dataset for a CONUS 12UTC 1 January 2003 test case from

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

Once you have downloaded the *wrfvar-testdata.tar.gz* file to *$DAT_DIR*, then extract it by typing

gunzip wrfvar-testdata.tar.gz

tar -xvf wrfvar-testdata.tar

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

rm wrfvar-testdata.tar

What next?

OK, now you have downloaded the necessary data, you are ready to preprocess the supplied observation file.

The 3D-Var Observation Preprocessor (3DVAR_OBSPROC)

By this stage, you have successfully downloaded the test data for the tutorial and are ready to run 3DVAR observation preprocessor (3DVAR_OBSPROC).

Accessing and compiling the 3DVAR_OBSPROC code

First, go to your working directory and download the 3DVAR_OBSPROC code from

http://www.mmm.ucar.edu/wrf/src/3DVAR_OBSPROC_v2.2beta.tar.gz.

Once you have it on your machine, type the following to unzip and untar it:

gunzip 3DVAR_OBSPROC_v2.2beta.tar.gz

tar -xvf 3DVAR_OBSPROC_v2.2beta.tar

After this, you should see a directory3DVAR_OBSPROC/ created in your working directory. If so, remove the .tar file

rm 3DVAR_OBSPROC_v2.2beta.tar

and cd to this directory:

cd 3DVAR_OBSPROC

Read the README file. To compile 3DVAR_OBSPROC, type

make

Once this is complete (a minute or less on most machines), you can check for the presence of the 3DVAR_OBSPROC executable by issuing the command (from the 3DVAR_OBSPROC directory)

ls-l src/3dvar_obs.exe

-rwxr-xr-x 1 mmm01 system 591184 May29_ 2003 src/3dvar_obs.exe*

Running3DVAR_OBSPROC

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

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

then edit ***namelist.3dvar_obs***.

In this tutorial, all you need to change is the full path to the observation file (**ob.little_r**) file, the grid dimensions (nestix=nestjx=45), and the resolution (dis=200[km]).

To run the 3DVAR_OBSPROC, type

3dvar_obs.exe >&! 3dvar_obs.out

Looking at 3DVAR_OBSPROC output.

Once the ***3dvar_obs.exe*** has completed normally, you will have an observation data file: ***obs_gts.3dvar***, which will be used as the input to WRF-Var. Before running WRF-Var, there is a utility to look at the data distribution for each type of observations.

1) **cd MAP_plot**;

2) Modify the shell script ***Map.csh*** to set the time window and full path of input observation file (obs_gts.3dvar);

3) Type

Map.csh

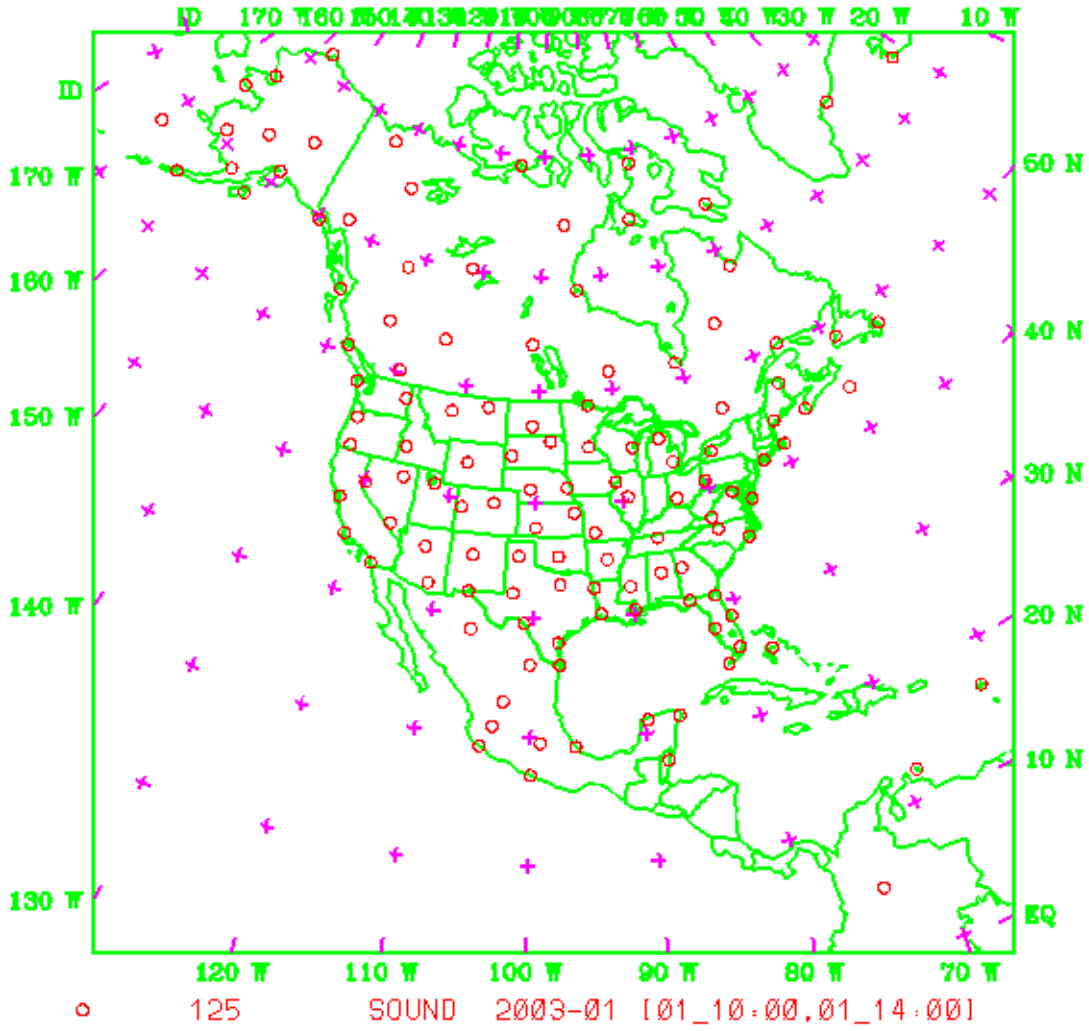
the *configure.user* will be automatically picked up for your computer system when you type ***Map.csh***;

4) When the job has completed, you will have a gmeta file ***gmeta.****{analysis_time}*

which 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.2003010112

The gmeta file illustrates the geographic distribution of sonde observations for this case. You should see something like the following:



Saving necessary file for 3DVAR and clean 3DVAR_OBSPROC

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

mvobs_gts.3dvar $*DAT_DIR*/obs_gts.3dvar.2003010112

Finally, to clean up the 3DVAR_OBSPROC directory, type

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 job failed;
4) In most cases, the job failed was caused by incorrect input files names, or the specified analysis time, time window, etc. 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 file3dvar_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 download and compile the WRF-Var code. A description of the script that controls the running of WRF-Var and an overview of the namelist that it creates is also given.

Accessing WRF-Var code

First, go to your working directory and download the WRF-Var code from

http://www.mmm.ucar.edu/wrf/src/wrfvar_v2.2beta.tar.gz

Once you have it on your machine, type the following to unzip and untar it:

gunzip wrfvar_v2.2beta.tar.gz

tar –xvf wrfvar_v2.2beta.tar

After this, you should see a program directory wrfvar/ created in your working directory. If so, tidy up with the command

rm wrfvar_v2.2beta.tar

cd to the wrfvar directory:

cd wrfvar_v2.2beta

and you should see something like the following files and subdirectories listed (may be slightly different depending on version used):

ls -l

```
total160

-rw-r--r--   1 dmbarker  users   15895 11 Jul 14:39 Makefile
-rw-r--r--   1 dmbarker  users   7189_ 4 Feb 07:40 README
-rw-r--r--   1 dmbarker  users   7263 11 Jul 14:39 README.NMM
-rw-r--r--   1 dmbarker  users   2548 18 Jan_ 2005README_test_cases
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 Registry
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 arch
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 chem
-rwxr-xr-x   1 dmbarker  users   1656 15 Apr 11:45 clean
-rwxr-xr-x   1 dmbarker  users  7225 25 Jul 09:48 compile
-rwxr-xr-x   1 dmbarker  users  10664 25 Jul 18:49 configure
drwxr-xr-x   7 dmbarker  users   1024 27 Jul 08:55 convertor
drwxr-xr-x   6 dmbarker  users   1024 27 Jul 08:55 da_3dvar
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 dyn_em
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 dyn_exp
drwxr-xr-x   3 dmbarker  users   2048 27 Jul 08:55 dyn_nmm
drwxr-xr-x  14 dmbarker  users   1024 27 Jul 08:55 external
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 frame
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 gen_be
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 inc
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 main
drwxr-xr-x   3 dmbarker  users   2048 27 Jul 08:55 phys
drwxr-xr-x   4 dmbarker  users   1024 27 Jul 08:56 run
drwxr-xr-x   3 dmbarker  users   1024 27 Jul 08:55 scripts
drwxr-xr-x   3 dmbarker  users   2048 27 Jul 08:55 share
drwxr-xr-x  12 dmbarker  users   1024 27 Jul 08:55 test
drwxr-xr-x   4 dmbarker  users   2048 27 Jul 08:55 tools
```

Compiling WRF-Var

To begin the compilation of WRF-Var, type the following to see the options available to you:

configure

You will then be queried with something like the following:

```
service01:>configure
option =
Please type : <configurevar> to compile var,
    or type : <configurebe> to compile be.
```

The two choices refer to creating compiler options for WRF-Var, or alternatively to begin setting up the calculation of background error statistics. Here, we wish to compile WRF-Var, and so type

configure var

You will then see a list of alternative compilation options, depending on whether you want to run single processor, with different compilers etc, e.g.

```
option = var
configure var
checking for perl5... no
checking for perl... found/usr/local/bin/perl (perl)
Will use NETCDF in dir:/usr/local/netcdf
PHDF5 not set inenvironment. Will configure WRF for use without.
option = var
configure for var

------------------------------------------------------------------

Please select from among the following supported platforms.



1.Settings for PC Linux i486 i586 i686, PGI compiler (Single-threaded)

2.Settings for PC Linux i486 i586 i686, PGI compiler (RSL, MPICH, RSL
IO)

3.Intel xeon i686 ia32 Xeon Linux, ifort compiler (single-threaded, no
nesting)


Enter selection [1-3] :
```

In the online tutorial, 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.defaults_wrfva r*in the wrfvar/arch subdirectory. This sets up compile options etc ready forcompilation. We recommend you run WRF-Var in single processor mode first, but you later want to run WRF-Var on distributed memory machines to really speed things up.

Check the WRF-Var configure file has been produced by typing

ls -l configure.wrf

Browse the file if interested, then to compile WRF-Var type

compile var

Once this is complete, you can check for the presence of the WRF-Var executable by issuing the command (from the wrfvar directory)

ls -l main/wrfvar.exe

```
-rwxr-xr-x  1 dmbarker users  11045892 May29  2003 main/wrfvar.exe
```

While you are waiting for compilation to finish, we suggest you read the following sections to acquaint yourself with how you are going to run WRF-Var.

The DA_Run_WRF-Var.csh script.

The WRF-Var system is run (do not do this yet in the online tutorial!) from the standard script *wrfvar/run/DA_Run_WRF-Var.csh*. An example ca be seen at

http://www.mmm.ucar.edu/wrf/WG4/wrfvar/DA_Run_WRF-Var.csh.txt

The script performs a number of tasks, as follows:

a) Specify job details via environment variables: The user should be able, in most cases, to run WRF-Var just by resetting default environment variables (set below) to values for your particular application. Examples include changing data directory, filenames, and particular parameters within WRF (namelist.3dvar) and WRF (namelist.input) files.
b) Define default environment variables. These can be overridden in a) by setting non-default values.
c) Perform sanity checks (e.g. ensure input files exist). d)Create 3D-Var namelist file *namelist.3dvar*
d) Prepare for assimilation: link, and copy data/executables to the run_ directory.
e) Create *namelist.3dvar* and *namelist.input* files from environments specified above.
f) Run WRF-Var.

A brief description of the 3D-Var namelist options is now given.

The WRF-Var namelist.3dvar file

There are many options you can change in a data assimilation system: withholding particular observations, empirically retuning background errors, changing the minimization convergence criteria, etc, etc. To see the current full list of namelist options, check out the section of DA_Run_WRF-Var.csh that sets the default values. Not all values are provided with comments. This is deliberate – we only support changing the values with comments! Feel free to experiment with the others only if you can support yourself by checking the code to see what these other options do!

The namelist.3dvar file created for the tutorial case can be seen at

http://www.mmm.ucar.edu/wrf/WG4/wrfvar/namelist.3dvar.txt

The WRF namelist.input

Since WRF-Var is now a core for the WRF model, it uses the WRF framework to define and perform parallel, I/O functions. This is fairly transparent in the WRF-Var code. However, one disadvantage is that WRF-Var now requires one to specify the grid dimensions at run-time via a fil e(the previous serial version of MM5 3DVAR would pick these dimensions up from the input file and dynamically allocate memory).

For this tutorial, a 45x45x28 200km resolution case-study is used. These grid dimensions are specified via the default environment variables in the DA_Run_WRF-Var.csh script. You will need to change these values for your own domains.

The namelist.input file created for the tutorial case can be seen at

http://www.mmm.ucar.edu/wrf/WG4/wrfvar/namelist.input.txt

What next?

Having compiled WRF-Var and familiarized yourself with the script and namelist files, it's time to run WRF-Var!

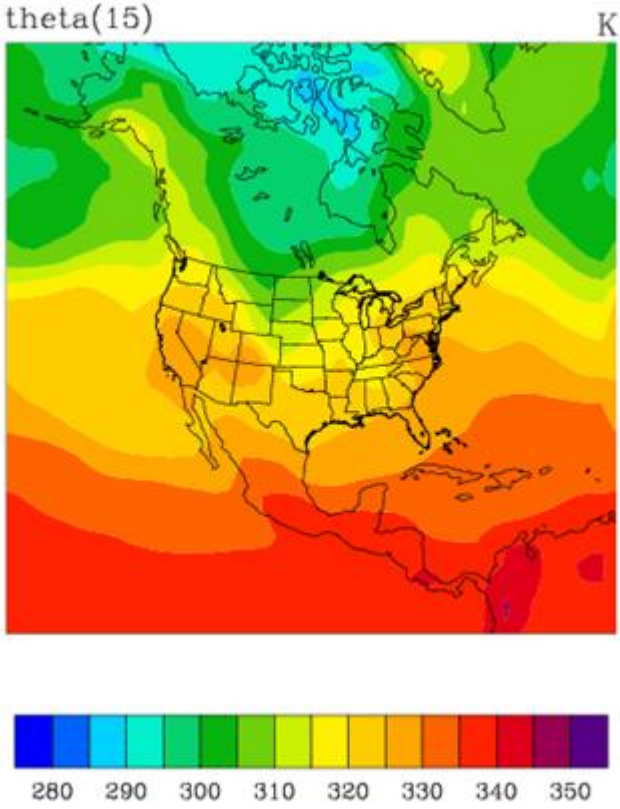
Run WRF-Var CONUS Case Study

In this section, you will learn how to run WRF-Var using observations and a first guess from a low-resolution (200km) 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. If not, then you will need to return to stages a. to c.

The Case

The data for this case is valid at 12 UTC 1st January 2003. The first guess comes from the NCEP global assimilation system (SSI), passed through the WRF *Standard Initialization (SI)* and *real* packages. The first-guess level 15 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). To simplify the system further, only radiosonde observations areas simulated.

Required3D-Var namelist changes

You will need to download the script DA_Run_WRF-Var.csh.txtand **USE IT TO OVERWRITE THE FILE DA_Run_WRF-Var.csh IN SUBDIRECTORY run**.. Open this script with your favorite editor, and spend some time browsing it to accustom yourself with the layout. Do not despair, in this tutorial we will not be changing too much in here. See c. Setting Up WRF-Var for more information on the various namelist options.

In your first experiment, the only changes you need to make to the *DA_Run_WRF-Var.csh* script are to specify the DAT_DIR (input data directory) and WRFVAR_DIR (WRF-Var code directory) and RUN_DIR (where the run happens) environment variables.

Run 3D-Var

Once you have set the necessary environment variables, run WRF-Var for this case study by typing

DA_Run_WRF-VAR.csh

in the wrfvar/run subdirectory. (ignore any messages about hosts files)

Successful completion of the job results in a number of output diagnostics files in the ${RUN_DIR}/wrf-var subdirectory 2 levels above. The 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 give yourself more experience with some of the 3D-Var options, try rerunning 3D-Var with different namelist.3dvaroptions. For example, making the WRF-Var convergence criteria more stringent by reducing the value of DA_EPS0 to e.g. 0.001. If WRF-Var has not converged by the maximum number of iterations (DA_NTMAX) then you may need to increase DA_NTMAX.

You may wish to change the RUN_DIR environment variable if you want to save all the outputs.

v. Whatnext?

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 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.

By this stage of the online tutorial you have successfully compiled and run WRF-Var.

Which are the important diagnostic 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 the previous case-study:

cd ${RUN_DIR}/wrf-var

ls -l

You will see something like the following:

```
total 37990
-rwxr-xr-x  1 dmbarker users  10816316 27 Jul 19:43wrfvar.exe
lrwxr-xr-x  1 dmbarker users        63 27 Jul 19:43wrf_3dvar_input
-rw-r--r--  1 dmbarker users      5508 27 Jul 19:43namelist.input.txt
lrwxr-xr-x  1 dmbarker users        64 27 Jul 19:43fort.92
lrwxr-xr-x  1 dmbarker users        47 27 Jul 19:43fort.33
-rw-r--r--  1 dmbarker users      2686 27 Jul 19:43namelist.3dvar.txt
-rw-r--r--  1 dmbarker users      8761 27 Jul 19:43LANDUSE.TBL
-rw-r--r--  1 dmbarker users       194 27 Jul 19:43fort.82
-rw-r--r--  1 dmbarker users       194 27 Jul 19:43fort.81
-rw-r--r--  1 dmbarker users     11848 27 Jul 19:43fort.12
-rw-r--r--  1 dmbarker users      5898 27 Jul 19:44wrfvar.out
-rw-r--r--  1 dmbarker users   4822312 27 Jul 19:44wrf_3dvar_output
-rw-r--r--  1 dmbarker users    158513 27 Jul 19:44fort.60
-rw-r--r--  1 dmbarker users   2827915 27 Jul 19:44fort.50
-rw-r--r--  1 dmbarker users       364 27 Jul 19:44fort.48
-rw-r--r--  1 dmbarker users    719125 27 Jul 19:44fort.47
-rw-r--r--  1 dmbarker users       194 27 Jul 19:44DAProg_WRF_Var.grad_fn
-rw-r--r--  1 dmbarker users     11848 27 Jul 19:44DAProg_WRF-Var.statistics
-rw-r--r--  1 dmbarker users       194 27 Jul 19:44DAProg_WRF-Var.cost_fn
```

The most important output files here are

*wrfvar.out:*Text file containing information output as WRF-Var is running. Again, there is a host of information on number of observations, minimization, timings etc.

DAProg_WRF-Var.statistics: Text file containing O-B, O-A 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 A-B statisticsi.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 grid.

fort.50: Data file containing detailed observation-based information for every observation, e.g. location, observed value, O-B, observation error, O-A, and quality control flag. This information is also used in a number of tuning programs found in the wrfvar/da_3dvar/utl directory.

wrf_3dvar_output: WRF (NETCDF) format file containing the analysis.

Take time to look through the textual output files to ensure you understand how 3DVAR has performed. For example,

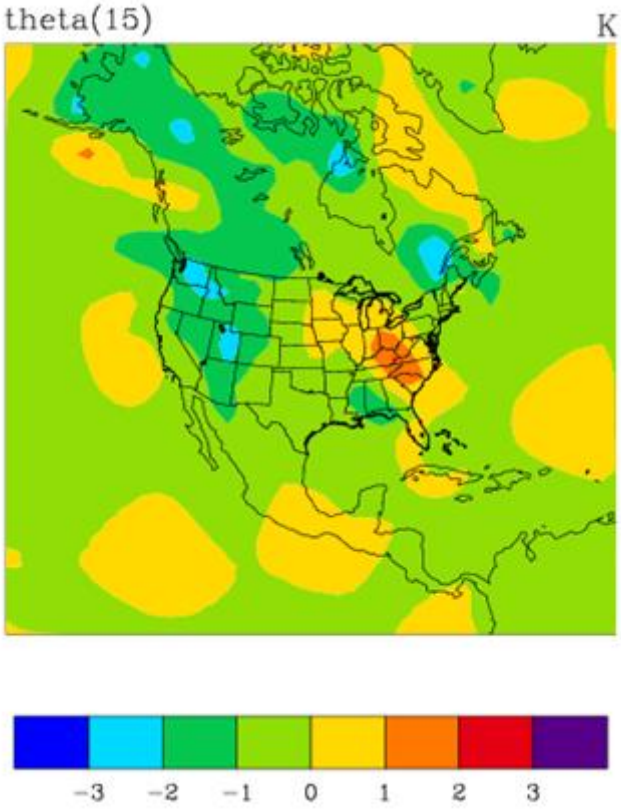
How closely has WRF-Var fitted individual observation types? - Look in the *DAProg_WRF-Var.statistics* file to compare the O-B and O-A statistics.

How big are the analysis increments? Again, look in the *DAProg_WRF-Var.statistics* file to see minimum/maximum values of A-B for each variable.

How long did WRF-Var take to converge? Is it converged? Look in the *wrfvar.out* file, which will indicate 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 (DA_NTMAX) then you may need to increase this value to ensure 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 15 theta increments below was produced using the particular NCL script wrftest.ncl. The plot indicates the magnitude, and scale of theta increments - the pattern is a function of observation distribution, observation minus first guess difference, and background error covariances used.



What next?

OK, you have run WRF-Var, checked out the diagnostics and are confident things are OK. Before running a forecast, you must first modify the tendencies within the lateral boundary condition files to be consistent with the new 3D-Var initial conditions.

Updating WRF lateral boundary conditions

Before running a forecast, you must first modify the tendencies within the lateral boundary condition files to be consistent with the new 3D-Var initial conditions. This is a simple procedure performed by the WRF utility WRF_BC.

Accessing the WRF_BC code

First, go to your working directory and download the WRF_BC code from

http://www.mmm.ucar.edu/wrf/src/WRF_BC_v2.1.tar.gz

Once you have it on your machine, type the following to unzip and untar it:

gunzipWRF_BC_V2.1.tar.gz

tar -xvf WRF_BC_V2.1.tar

After this, you should see a program directory WRF_BC_V2.1/created in your working directory. If so, tidy up with the command

rm WRF_BC_V2.1.tar

cd to the WRF_BC directory:

cd WRF_BC_V2.1

Compiling WRF_BC

The README file contains information on the role of WRF_BC_V2.1, and how to compile. To do the latter, simply type

make

You will then see an executable by typing

ls -l update_wrf_bc.exe

Running WRF_BC

The update_wrf_bc.exe is run via the new_bc.csh script. Simply supply the names of the input analysis, lateral boundary condition, and WRF-Var analysis file names. The output can then be used as the initial conditions for WRF.

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. Good luck!

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 7: WRF Software

Table of Contents

- Introduction
- WRF Build Mechanism
- Registry
- I/O Applications Program Interface (I/O API)
- Timekeeping
- Software Documentation
- Portability and Performance

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.

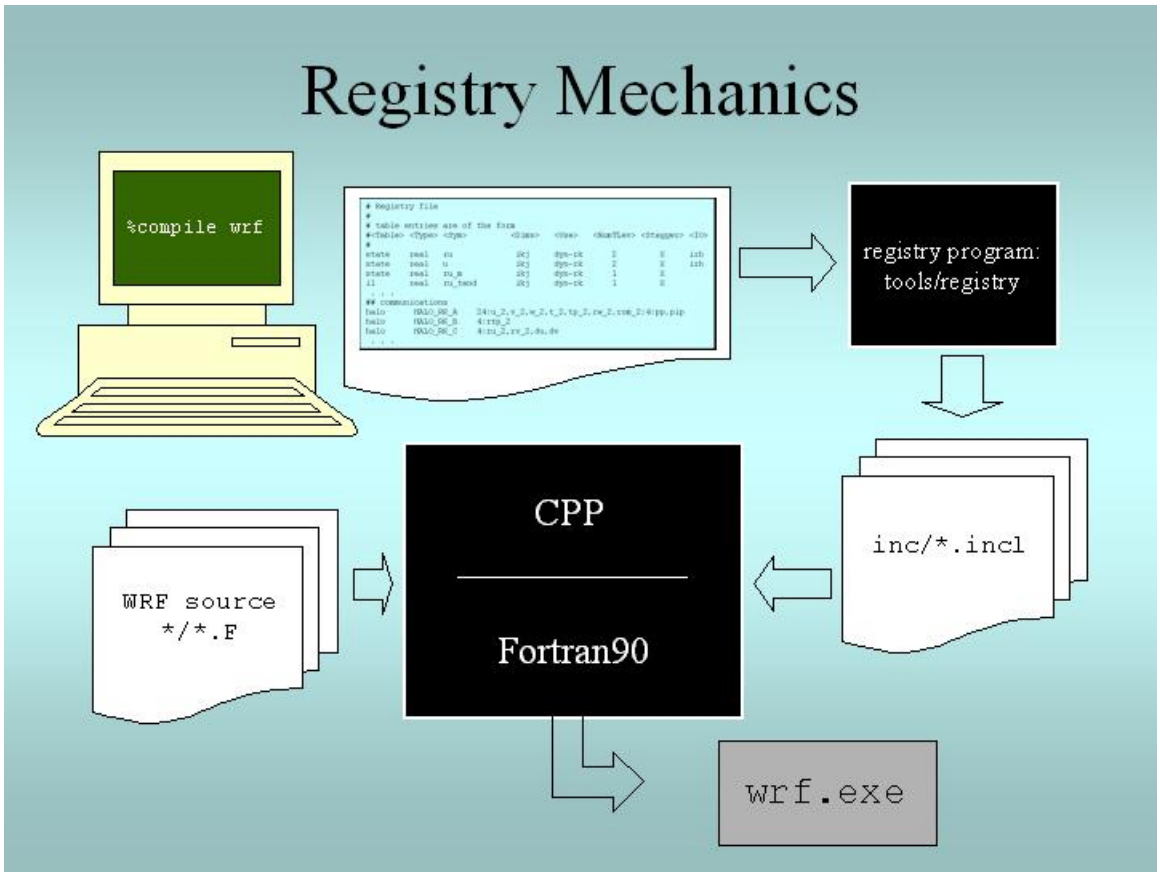


Figure 1. When the user compiles WRF, the Registry Program reads Registry/Registry, producing auto-generated sections of code that are stored in files in the inc directory. These are included into WRF using the CPP preprocessor and the Fortran compiler.

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

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 8: Post-Processing Utilities

Table of Contents

- Introduction
- NCL
- RIP4
- ARWpost
- read_wrf_nc utility
- iowrf utility

Introduction

There are a number of visualization tools available to display 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 four post-processing utilities are supported, NCL, RIP4, WRF2GrADS and WRF2VIS5D. All these programs can only read ARW data in netCDF 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*). The ARW post-processing packages assume that the data from the ARW model is using the netCDF libraries.

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/*), requires the NCAR Command Language written by NCAR Scientific Computing Division
- RIP (*http://www.atmos.washington.edu/~stoeling/*), requires NCAR Graphics
- GrADS (*http://grads.iges.org/home.html*), requires the GrADS visualization software
- Vis5D (*http://www.ssec.wisc.edu/~billh/vis5d.html*), requires the Vis5D visualization software

NCL

Ready-made NCL scripts are provided to create meta files for both real and ideal datasets. These scripts are relatively easy to read and change to generate different or more plots.

These notes describe the September 2005 release. This release requires high-level NCL scripts to run. This release can process ARW static/input and output files, as well as WRF-Var output data. This package can process both single and double precision data.

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, please visit: *http://www.ncl.ucar.edu/overview.shtml*

Necessary software

- Obtain the WRF_NCL TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
- NCAR Command Language libraries (*http://www.ncl.ucar.edu/*). This WRF_NCL beta release requires at least NCL version 4.2.0.a032 to run.

Hardware

The code has been tested on the following machines

- DEC Alpha
- IBM
- Linux (pgi and intel compilers)
- MAC

The code should run equally well on other machines, but has not been tested on all platforms.

Steps to compile and run

Untar WRF_NCL TAR file

Inside the TAR file, you will have the following files:

DOThluresfile	*Control plotting background* ***MUST be copied to*** **~/. hluresfile** ***before any scripts are run***
README_FIRST README_NCL gsn_code.ncl SkewTFunc.ncl WRFOptions.ncl WRFPlot.ncl WRFUserARW.ncl	*Readme Files and* *NCL function scripts*
wrf_user_fortran_util_0.f make_ncl_fortran make_ncl_fortran.alpha make_ncl_fortran.ibm make_ncl_fortran.ibm64 make_ncl_fortran.linux make_ncl_fortran.sun	*FORTRAN utility file and* *make files to compile the utility program*
wrf_bwave.ncl wrf_grav2d.ncl wrf_hill2d.ncl wrf_qss.ncl wrf_squall_2d_x.ncl wrf_squall_2d_y.ncl wrf_real_input.ncl wrf_real.ncl wrf_cloud.ncl	*NCL scripts for ideal and real data*

FIRST

Copy the file **DOThluresfile** to **~/.hluresfile**

This file controls the color / background / fonts and basic size of your plot. This file MUST have the name .hluresfile, and MUST be located in your home directory.

For more information regarding this file, see:
http://www.ncl.ucar.edu/Document/Graphics/hlures.shtml

Build the external function, wrf_user_fortran_util_0.so

- NCL has the ability to link in FORTRAN shareable object files. This provides an easy way to compute diagnostic quantities and to performing interpolations.

- Presently, only one FORTRAN object needs to be built -
  wrf_user_fortran_util_0.so
- To build the FORTRAN object, run the "**WRAPIT**" script provided by NCL:
  e.g. **WRAPIT wrf_user_fortran_util_0.f**
  HINT: you may need to find the script and use the full path to this script.
- If successful, you will see the following file created in your WRF_NCL directory:
  wrf_user_fortran_util_0.so

- Under rare occasions, this script may not work. In this case, try one of the **make_ncl_fortran** csh scripts provided in the WRF_NCL directory.
  e.g. **make_ncl_fortran wrf_user_fortran_util_0**
  ***HINT*:** The most common error when building the external function, is not finding the "wrapit77" function on your system. "wrapit77" is part of the NCAR Graphics routines. If you run into this problem, make sure your path to this function is setup correctly. *(If all else fails, locate the file on your system and copy it to your WRF_NCL directory.*
- If successful, you will see these files created in your WRF_NCL directory:
  so_locations
  wrf_user_fortran_util_0.o
  wrf_user_fortran_util_0.so

- For more information about **WRAPIT** and **wrapit77**, see:
  http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclExtend.shtml

Edit the script you want to run

- Change the location and name of the file:
  a = addfile("../../WRFV2/run/wrfout_d01_2000-01-24_00:00:00.nc","r")
- Do not remove the ".nc" after the file name - the script needs it to determine which type of input file (netCDF) it is dealing with *(Note the .nc is NOT part of the actual file name.)*
- Specify the type of plot you want to generate:
  type = ……
  Options are x11, pdf, ps, ncgm
- Specify the output file name:
  wks = gsn_open_wks(type,"**MyOutput**")
- Give your output a title:
  res@MainTitle = "REAL-TIME ARW"

Control plotting options with WRFOptions.ncl

MainTitlePos	Options are top Left, Center, Right
InitTime	Add Initial Time to plot
ValidTime	Add Valid Time to Plot Switch off for files which do not have a valid time
TimePos	Placement of Time Information. Options are top Left or Right. If MainTitlePos and TimePos are on same side, Time Information will be plotted under the Main Title
Footer	Add some model information as a Footer to the plots
mpOutlineBoundarySets	Default is *GeophysicalandUSStates* Alternatively set to *AllBoundaries*

Basic flow of an NCL script

```
load "WRFOptions.ncl"   ; set basic plot options here
load "gsn_code.ncl"     ; high-level NCL scripts
load "WRFPlot.ncl"      ; WRF functions / procedures
load "WRFUserARW.ncl"   ; Diagnostics

begin
  a = addfile("wrfout_d01_2000-01-24_12:00:00.nc","r") ; open file
  type = "pdf"    ; Will create pdf output files
  wks = gsn_open_wks(type,"wrfout") ; open workstation + set output

  res@MainTitle    = "ARW Real Data"  ; Set plot title

  times  = wrf_user_list_times(a)  ; get times in the file
  ntimes = dimsizes(times)         ; number of times in the file

  do it = 0,ntimes-1          ; loop through file for all times

    res@TimeLabel  = times(it)      ; Set Valid time info

    map = wrf_map(wks,a,res)        ; Create a map background

    ter = wrf_user_getvar(a,"HGT",time)   ; get terrain from file
    t2  = wrf_user_getvar(a,"T2",time)    ; get T2 from file

   ; Set options and create terrain plot
    opts_ter = res
    opts_ter@cnFillOn           = True
    contour_ter  = wrf_contour(a,wks,ter,opts_ter)
```

```
  ; Set options and create t2 plot
    opts_t2 = res
    contour_t2  = wrf_contour(a,wks,t2,opts_t2)

   ; Overplay terrain and t2 plots and place on map background
    wrf_map_overlay(wks,map,(/contour_ter, contour_t2/),False)
  end do

end
```

Run the NCL script

- To run the script, type:
  ncl < NCL_script (or "ncl NCL_script" for higher versions of NCL)
  e.g. **ncl < wrf_real.ncl**
- This will create an output file
  e.g. MyOutput.(pdf/ps/ncgm)
  If type of output was set to "x11", no hardcopy will be produced.
- The name of the graphics file that is created, is controlled by the line:
  wks = gsn_open_wk (type,"MyOutput") inside the NCL script

View the output file

- To view a meta file *(type = ncgm)*, use the command "idt", e.g.
  idt MyOutputs.ncgms
- Examples of plots created for both idealized and real cases are available from:
  http://www.mmm.ucar.edu/wrf/users/graphics/WRF_NCL/NCL.htm

Miscellaneous

For print quality images, create pdf or ps images directly via the ncl scripts (type = pdf / ps).

To convert NCGM files to GIF images, a very handy tool is the ncgm2gif script (*http://ngwww.ucar.edu/info/ncgm2gif*)

To run the script, type:
ncgm2gif ***metafile***
e.g. ncgm2gif -res 500x500 -nomerge test.cgm

This will convert all images in test.cgm to 500x500 pixel gif images, called testxxx.gif

A compete list of options are available inside the ncgm2gif script (*http://ngwww.ucar.edu/info/ncgm2gif*)

RIP4

RIP4 was adapted from the RIP code, originally developed to display MM5 model data. *(Primarily Mark Stoelinga, from both NCAR and the University of Washington developed RIP).*

The code reads ARW (and MM5) output files and creates meta file plots.
Since version 4.1 RIP4 can read both real and idealized ARW datasets.
Since version 4.2 RIP4 can also read all the files produced by WPS.

The RIP **users' guide** (*http://www.mmm.ucar.edu/wrf/users/docs/ripug.htm*) is essential reading.

Necessary software

- Obtain the RIP4 TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
- NCAR Graphics software (*http://ngwww.ucar.edu/*)

Hardware

The code has been ported to the following machines

- DEC Alpha
- Linux (pgi and intel compilers)
- MAC (xlf and absoft compilers)
- SUN
- SGI
- IBM
- CRAY
- Fujitsu

Steps to compile and run

Untar RIP4.TAR.gz file

Inside the TAR file you will see the following files:

CHANGES
Doc/
Makefile
README
color.tbl
psadilookup.dat
rip_sample.in
ripdp_sample.in

src/
stationlist
tabdiag_sample.in
tserstn.dat
bwave.in | *new in version 4.1*
custom_maps/ | *new in version 4.1*
grav2d_x.in | *new in version 4.1*
hill2d.in | *new in version 4.1*
qss.in | *new in version 4.1*
sqx.in | *new in version 4.1*
sqy.in | *new in version 4.1*

Compile the code

Typing "make" will produce the following list of compile options

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 mac_xlf	*To Run on MAC_OS_X with Xlf Compiler*
make mac	*To Run on MAC_OS_X with Absoft Compiler*
make sun	*To Run on SUN*
make sun2	*To Run on SUN if make sun didn't work*
make sun90	*To Run on SUN usingF90*
make sgi	*To Run on SGI*
make sgi64	*To Run on 64-bit SGI*
make ibm	*To Run on IBM SP2*
make cray	*To Run on NCAR's Cray*
make vpp300	*To Run on Fujitsu VPP 300*
make vpp5000	*To Run on Fujitsu VPP 5000*
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 dec* will compile the code for a DEC Alpha computer

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_wrf.

ripcomp	This program reads in two rip data files and compares their contents.
ripdp_mm5	RIP Data Preparation program for MM5 input data
ripdp_wrf	RIP Data Preparation program for WRF input 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.

Prepare the data

- To prepare the data for the RIP program, one much first run RIPDP (RIP Data Preparation), for WRF
- As this step will create a large number of extra file, creating a new directory to place these files in, will enable you to manage the files easier
  mkdir DATA
- Edit the namelist **ripdp_sample.in** *(the use of this namelist is optional)*
  The most important information needed in the namelist, is the times you want to process
- Run ripdp for WRF
  ripdp_wrf [-n namelist_file] casename [basic|all] data_file_1 data_file_2 data_file_3 ...
  e.g**. ripdp_wrf -n ripdp_sample.in DATA/real basic ../DATA/real/wrfout_d01_2000-01-24_12:00:00 >& ripdp_log**

Create graphics - step 1

- The first step in creating the graphics you are interested in, is to edit the User Input File (UIP) rip_sample.in *(or create your own UIP)*

- The UIP file, consists of
  - 2 namelists **userin** (which control the general input specifications) and trajcalc (which control the creation of trajectories); and
  - the Plot Specification Table (**PST**), used to control the generation of the graphics

- namelist: **userin**

Namelist Variable	**Variable Type**	**Description**
idotitle	Integer	Control first part of title line.
titlecolor	Character	Control color of the title lines
ptimes	Integer	Times to process. This can be a string of times or a series in the form of *A,-B,C,* which means "times from hour *A*, to hour *B*, every *C* hours"
ptimeunits	Character	Time units. This can be either `h' (hours), `m' (minutes), or `s' (seconds)
tacc	Real	Time tolerance in seconds. Any time in the model output that is within *tacc* seconds of the time specified in *ptimes* will be processed.
timezone	Integer	Specifies the offset from Greenwich time.
iusdaylightrule	Integer	Flag to determine if US daylight saving is applied.
iinittime	Integer	Controls the plotting of the initial time on the plots.
ivalidtime	Integer	Controls the plotting of the plot valid time.
inearesth	Integer	Plot time as two digits rather than 4 digits.
flmin	Real	Left frame limit
flmax	Real	Right frame limit
fbmin	Real	Bottom frame limit
ftmax	Real	Top frame limit
ntextq	Integer	Quality of the text
ntextcd	Integer	Text font
fcoffset	Integer	Change initial time to something other than output initial time.

idotser	Integer	Generate time series output files (no plots) only ASCII file that can be used as input to a plotting program).
idescriptive	Integer	Use more descriptive plot titles.
icgmsplit	Integer	Split metacode into several files.
maxfld	Integer	Reserve memory for RIP.
ittrajcalc	Integer	Generate trajectory output files (use namelist *trajcalc* when this is set).
imakev5d	Integer	Generate output for Vis5D

- 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=850,700,-300,100; >
        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 overlaid plots:

– Temperature in degrees C (feld=tmc). This will be plotted as a horizontal contour plot (ptyp=hc), on pressure levels (vcor=p). The pressure levels used will be 850 and 700 to 300 in steps of 100 mb (thus 5 plots will be generated, on 850, 700, 600, 500, 400, and 300 mb). 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*).

Create graphics - step 2

- First set the environment variable:
  setenv RIP_ROOT *your_rip4_directory*
- Run rip
  rip [-f] model_case_name rip_case_name
  e.g. **rip -f DATA/real rip_sample**
- If this is successful, the following files will be created
  rip_sample.cgm *gmeta file*
  rip_sample.out *log file - view this file if a problem occurred*

View the meta file

- To view the meta file, use the command "idt", e.g.
  idt rip_sample.cgm
  Examples of plots created for both idealized and real cases are available from:
  http://www.mmm.ucar.edu/wrf/users/graphics/RIP4/RIP4.htm

Miscellaneous

To convert NCGM files to GIF images, a very handy tool is the ncgm2gif script (*http://ngwww.ucar.edu/info/ncgm2gif*)

To run the script, type:
ncgm2gif ***metafile***
e.g. ncgm2gif -res 500x500 -nomerge test.cgm

This will convert all images in test.cgm to 500x500 pixel gif images, called testxxx.gif

A compete list of options are available inside the ncgm2gif script (*http://ngwww.ucar.edu/info/ncgm2gif*)

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

- Obtain the ARWpost TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
- WRFV2 must be installed and available somewhere, as ARWpost makes use of the common IO API libraries from WRFV2.
- 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.

Hardware

The code has been ported to the following machines

- DEC Alpha
- Linux (pgf and intel compilers)
- MAC (with xlf compilers)
- SUN
- SGI
- IBM

Steps to compile and run

Untar ARWpost TAR file

Inside the TAR file you will have the following files:

README	
arch/	*Configure and compilation control*

clean* compile* configure*	
namelist.ARWpost	*namelist*
fields.plt myLIST	*Samples to illustrate how to add fields to plot via a file*
src/	*Source code*
scripts/	*Sample scripts*
gribinfo.txt gribmap.txt	*Files needed to process GRIB1 data. Do not edit these files*
util/	*Utility program*

Configure

- WRFV2 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

- If your WRFV2 code is NOT compiled under ../WRFV2, edit configure.arwp, and set "**WRF_DIR**" to the correct location of your WRFV2 code.
- Type: ./compile
- This will create the ARWpost.exe executable

Edit the *namelist.ARWpost* file

- Set times to process (***&datetime***)
- Set input and output file names and fields to process (***&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. Do not use wild characters in the *input_root_name*.
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.
plot	Which fields to process. Options are: all, basic, list, file, basic_file, basic_list, list_file, all_file, all_list, basic_list_file, all_list_file 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*" If "*file*" is used, a list of variables must be added to a file, and the filename supplied under "*fields_file*"
fields	Fields to plot. Only used is *list* was used in the "*plot*" variable.
fields_file	File name that contains list of fields to plot. Only used is *file* was used in the "*plot*" variable.
output_type	Options are 'grads' (default) or 'v5d'
mercator_defs	Set to true if mercator plots are distorted

- *Available diagnostics*: **height** (model height in km), **theta** (potential temperature), **tc** (temperature in degrees C), **tk** (temperature in degrees K), **td** (dew point temperature in degrees C), **rh** (relative humidity), **pressure** (full model pressure in hPa), **dbz** (3d reflectivity), **max_dbz** (maximum reflectivity), **slp** (sea level pressure), **mcape** (maximum CAPE), **mcin** (maximum CIN), **lcl** (lifting condensation level), **lfc** (level of free convection), **cape** (3d CAPE), **cin** (3d CIN), **umet** and **vmet** (winds rotated to earth coordinates), **u10m** and **v10m** (10m winds rotated to earth coordinates), **wdir** (wind direction), **wspd** (wind speed)

- Set levels to interpolate too (***&interp***)

  interp_method: 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

- For extra information regarding the namelist.ARWpost file, refer to the README file.

Run

- 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 script 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

plotALL.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*

read_wrf_nc utility

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.

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 has been ported to Dec Alpha, Linux, Mac (running xlf compiler), Sun, SGI and IBM

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

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 xy 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 variables will be created. -ts xy X Y VAR VAR ….. will generate time series output for all VAR's at location X/Y -ts xy 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:

```
elseif ( var == 'U') then
  data_real = 10.0
```

  Example 2:
  If you want to change a section of the LANDMASK data to SEA points:

```
elseif ( 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):

```
elseif ( var == 'ISLTYP') then
  where (data_int == 3 )
    data_int = 7
  endwhere
```

- 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

iowrf utility

This utility allows a user to do some basic manipulation on WRF ARW netCDf files.

- The utility allow one to thin the data, or extract a box from a 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 has been ported to Dec Alpha, Linux, Mac (running xlf compiler), Sun, SGI and IBM

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

This will create the executable: **iowrf**

Run

iowrf wrf_data_file_name [-options]

options : [-h / help]
[-thina X] [-thin X] [-box {}]
[-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
-64bit	Allow large files (> 2GB) to be read / write

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Appendix A: WRF Standard Initialization - Preparing Input Data

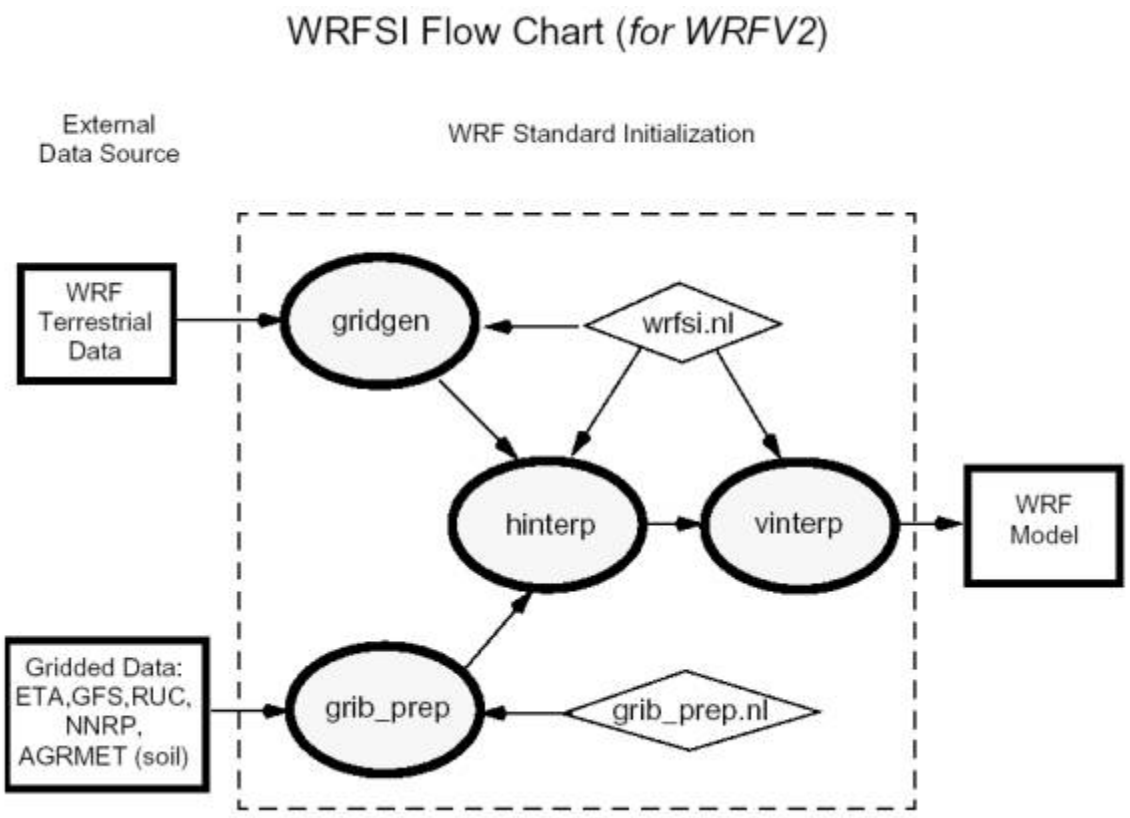
Table of Contents

- Introduction
- Function of Each SI Program
- How to Install WRFSI?
- How to Run WRFSI?
- WRFSI GUI
- Using WRFSI For Nesting
- Using Multiple Data Sources
- Checking WRFSI Output
- Description of Namelist Variables
- List of Fields in WRFSI Output

Introduction

The WRFSI has been replaced by the WRF Preprocessor System (WPS). Documentation of the new WPS is available in chapter 3 of this document.

The WRF Standard Initialization (WRFSI) is the first step to set up the model for real-data simulations. The software is a collection of four programs that together provides the input data required by the WRF model to start a real-data simulation. The following figure illustrates the program components and data flow in WRFSI:



The WRFSI program takes a user's definition of a domain (or domains for a nested run), together with various terrestrial datasets for terrain, landuse, soil type, annual deep soil temperature, monthly vegetation fraction, maximum snow albedo, monthly albedo, slope data, and meteorological data from another model (in GriB format) to create mesoscale domain, and interpolate the above data to this domain. The output from WRFSI is in netCDF format and it is conforming to WRF I/O API.

The WRFSI program has been successfully ported to a number of Unix-based machines. These include Compaq Alpha, IBM, Linux (using both PGI and Intel compiler), SGI Altix, AMD Opteron (PGI compiler only). Makefiles are also available for Sun and SGI, but limited tests have been performed.

The WRFSI code runs on single processor machines only. The code is memory efficient.

Function of Each SI Program

The WRFSI program consists of four major independent programs: ***grib_prep***, ***gridgen_model***, ***hinterp*** and ***vinterp***. It also has a few utility programs, ***siscan***, ***staticpost***, and ***plotfmt***.

Program *gridgen_model*

The function of the program *gridgen_model* is to define a simulation domain, and read and interpolate various terrestrial datasets from latitude/longitude grid to the projection grid. The simulation domain is defined based on information specified by the user in SI's namelist file, section `'hgridspec'` in wrfsi.nl. The terrestrial inputs that *gridgen_model* uses include terrrain, landuse, soil type, annual deep soil temperature, monthly vegetation fraction, maximum snow albedo, monthly albedo, and slope data. These data are privided from WRF Users' Web site: http://www.mmm.ucar.edu/wrf/users/. WRFSI supports three projection types: Lambert-Conformal, Polar stereographic, and Mercator.

Program *grib_prep*

The function of the program *grib_prep* is to read GriB files, degrib the data, and write the data out in a simple format, which is referred to as the intermediate format. The GriB files contain time-varying meteorological fields and are typically from another regional, or global model, such as NCEP's NAM (or Eta), and GFS models. SI supports the GriB format Edition 1 on many platforms, but only supports GriB 2 on 32-bit Linux at this time.

Each GriB dataset may contain data more than we need to initialize WRF model. To limit the data we require from the GriB files, Vtables are employed by which GriB code, and level codes are used to identify a particular field. Different GriB files may have different codes for the same variable, hence different Vtables are prepared for commonly available GriB files.

The provided Vtables are available for NAM/Eta 104, 212 grids, NAM/Eta data in AWIP format, GFS (or AVN), NCEP/NCAR Reanalysis archived at NCAR, RUC (pressure level data and hybrid coordinate data), and AFWA's AGRMET land surface model output.

If you have a GriB dataset that you would like to use to start the model, follow the Vtable examples given in the SI tar file under directory `extdata/static/` and create one for your dataset.

You can also take advantage of the intermediate format to ingest any data you may have as long as they are on pressure levels (data on other coordinate will require code modifications). A description of the intermediate format can be found on http://wrfsi.noaa.gov/, or README file in the SI program tar file (section 3.2.1).

Program *hinterp*

The function of the program *hinterp* is to horizontally interpolate meteorological data degribbed by the *grib_prep* program onto the simulation domain created by *gridgen_model*. The methods of horizontal interpolation may be controlled by namelist variables

Program *vinterp*

The function of the program *vinterp* is to vertically interpolate meteorological data from pressure (or hybrid data in the case of RUC) levels to WRF's eta coordinate, which is defined by the user in the SI's namelist section `'interp_control'` in *wrfsi.nl*.

Utility Program *siscan*

Program siscan is a utility program which may be used to read hinterp output (file name begins with hinterp.d01.*).

```
     siscan hinterp.d01.2000-01-24_12:00:00

 Scanning hinterp.d01.2000-01-24_12:00:00
 Domain Metadata Information
 ---------------------------------------------------
Domain Number .........  1
Parent ID ............. -1
Dynamic Init. Source .. SI
Static Init. Source ... SI
Valid Date (YYYDDD) ... 2000024
Valid Time (sec UTC) .. 43200.0
Origin X in Parent ....    -1
Origin Y in Parent ....    -1
Nest Ratio to Parent ..    -1
Delta X ............... 30000.0
Delta Y ............... 30000.0
Top Level .............   5000.0
Origin Z in Parent ....    -1
X dimension ...........    74
Y dimension ...........    61
Z dimension ...........    27
  ---------------------------------------------------
 Variables found:
 NAME    S D   NX   NY   NZ     UNITS     DESCRIPTION       MINVAL   MAXVAL   AVGVAL
 ------- - - ----- ----- --- ------------ ---------------- -------- -------- --------
PRESSURE 0 1    27     0   0     Pa       Pressure levels     5000.  200100.   59078.
T        4 3    74    61  27     K        Temperature       204.627  296.902  249.544
U        4 3    74    61  27     m s-1    U                -11.2302  57.4033  14.6849
V        4 3    74    61  27     m s-1    V                -52.7882  66.4211   5.9056
RH       4 3    74    61  27     %        Relative Humidit    1.000  117.639   52.779
SPECHUMD 4 3    74    61  27     kg kg-1  Specific Humidit 0.000000 0.016359 0.002082
HGT      4 3    74    61  27     m        Height                 0.   20601.    6206.
PMSL     4 2    74    61   0     Pa       Sea-level Pressu  100689.  102797.  101645.
PSFC     4 2    74    61   0     Pa       Surface Pressure   89520.  102235.  100210.
SNOW     1 2    74    61   0     kg m-2   Water Equivalent    0.000  184.144    5.537
SKINTEMP 1 2    74    61   0     K        Sea-Surface Temp  243.144  297.855  279.444
ST000010 1 2    74    61   0     K        T of 0-10 cm gro    0.000  291.070  143.554
ST010040 1 2    74    61   0     K        T of 10-40 cm gr    0.000  291.646  145.186
ST040100 1 2    74    61   0     K        T of 40-100 cm g    0.000  292.658  146.741
ST100200 1 2    74    61   0     K        T of 100-200 cm     0.000  294.566  148.923
SM000010 1 2    74    61   0              Soil Moisture of 0.000000 0.788069 0.162016
SM010040 1 2    74    61   0              Soil Moisture of 0.000000 0.787242 0.160085
SM040100 1 2    74    61   0              Soil Moisture of 0.000000 0.785682 0.156359
SM100200 1 2    74    61   0              Soil Moisture of 0.000000 0.782386 0.153110
SEAICE   1 2    74    61   0              Ice flag         0.000000 0.350000 0.000902
CANWAT   1 2    74    61   0     kg m-2   Plant Canopy Sur 0.000000 0.500000 0.194222
SOILHGT  1 2    74    61   1     m        Terrain height o -9999.00  1035.11 -4621.44
 End of file reached.
```

Utility Program *staticpost*

Utility program *staticpost* turns static.wrfsi.d0X files to WRF I/O API-conforming netCDF files, `wrfstatic_d0X`. This program is executed when running perl script *window_domain_rt.pl* (which also executes program *gridgen_model*). The `wrfstatic_d0X` files are not yet used by WRF model, but they will be used later for a simplier way to run two-way and one-way nesting.

Utility Program *plotfmt*

Utility program *plotfmt* can be used to make simple graphics from the intermediate files. It plots every field in the intermediate formatted data file. This utility is not automatically built when SI is installed. To compile this program, cd to *src/grib_prep/util* directory and type:

```
make plotfmt.exe
```

To run it, type

```
plotfmt intermediate-file-name
```

How to Install WRFSI?

The WRFSI program may be downloaded from http://wrfsi.noaa.gov/ page. There is a 'Installation README' file posted on the site. In this section, a summary of the installation procedure is provided.

Required Compilers, Scripting Language and Libraries

WRFSI code is written mostly in Fortran 77 and Fortran 90. A few utility programs are written in c. Hence, a Fortran 90 compiler, and C compiler (gcc is recommended) are required. These are the same requirement for WRF model. Perl scripts are used to run the SI program, and perl/Tk is required to run the GUI. WRFSI writes output in both binary (from *grib_prep* and *hinterp*) and netCDF (from *gridgen_model* and *vinterp*). A pre-installed netCDF library is required (again this is the same requirement as for WRF model).

Hint: Using PGI or Intel compiler on a **Linux** computer requires that the netCDF library is also installed using the same compiler.

Installation Steps

- Download the program tar file, type '`gunzip wrfsi_v2.1.tar.gz`' to unzip the file, and '`tar -xf wrfsi_v2.1.tar`' to untar the file. This will create a directory called wrfsi/.This will be the `SOURCE_ROOT` directory. If you do a 'ls -l' in this directory, you will see

  `CHANGES:` description of changes
  `HOW_TO_RUN.txt:` useful if you run SI not using the GUI
  `INSTALL:` instructions on how to install SI
  `README:` documentation of SI
  `README.wrfsi.nl:` description of namelist variables
  `Makefile:` top-level makefile
  `extdata/:` where you might want to place the degribbed data files
  `data/:` where the default directory and namelist file reside
  `src/:` the source code directory
  `graphics/:` directory where NCL scripts reside - may be use to make plots of *gridgen_model* output (*static.wrfsi.d0X* file)
  `gui/:` source directory for the GUI
  `util/:`

- Decide where you would like to place the executables and perl scripts that run various SI programs. This directory will be the INSTALLROOT. Also decide

  - where you would like to place the terrestrial datasets (terrain, landuse, etc.): GEOG_DATAROOT. Download the data from http://www.mmm.ucar.edu/wrf/users/download/get_source.html into this directory.
  - where you would place the intermediate formatted data files: EXT_DATAROOT
  - where you would like to run your case: DATAROOT and MOAD_DATAROOT. DATAROOT can be the top directory which contains multiple subdirectories, each of which is a MOAD_DATAROOT directory. If there is only one MOAD_DATAROOT, then MOAD_DATAROOT can also be the same as the DATAROOT directory.
  - where you would like the template directory to be. This directory will contain the SI namelist file that you would want to modify to create you own case. This is only relevant if you run SI not using the GUI.

- Once you have decided these, set the following environment variables:

```
setenv SOURCE_ROOT the-source-root-directory
setenv INSATLLROOT the-install-root-directory
setenv GEOG_DATAROOT where-terrestrial-data-are
setenv EXT_DATAROOT where-degribbed-files-are
```

```
setenv DATAROOT where-all-case-directory-is
setenv MOAD_DATAROOT where-one-case-directory-is
setenv TEMPLATES where-the-template-directory-is
```

Settng the environment variables can help one understand where things are. Note that the directory where the input GriB files reside are not defined through the environment variable. It is defined in the namelist file that *grib_prep* program uses. If you don't set these environment variables, the default environment variables are:

```
setenv SOURCE_ROOT /user-path/wrfsi
setenv INSTALLROOT /user-path/wrfsi
setenv GEOG_DATAROOT $INSTALLROOT/extdata
setenv EXT_DATAROOT $INSTALLROOT/extdata
setenv DATAROOT $INSTALLROOT/domains
setenv TEMPLATES $INSTALLROOT/templates
```

Whether you define these environment variables or use the default, you can find them in file, *config_paths*, in the $INSTALLROOT for later reference.

Hint: Do not use wrfsi/data/ directory for $DATAROOT.

Hint: Setting these environment variables correctly is critical every time you run the SI program. The run scripts and source code key on these environment variables to access data.

- The other environment variable to set is the location of the netCDF library:

```
setenv NETCDF /usr/local/netcdf
```

Once these environment variables are set, you are ready to run the install script: `install_wrfsi.pl` which resides in the top-directory of wrfsi. Type

```
perl install_wrfsi.pl
```

One will be prompted to answer whether you would like to install the GUI.

One may use command line options to specify netCDF path, machine type, whether to install gui, etc.. A typical command would look like:

```
Perl install_wrfsi.pl --path_to_netcdf --machine_type=pcintel \
                      --install_ui=n
```

where `machine_type` corresponds to the string in the provided makefiles in `src/include/makefile_{`*`machine_type`*`}.inc.in`. One may use this mechanism to build a makefile for a computer that is not yet supported.

If all the system utilities are in the right place and in your path (defined in your system resource file), the compiler, cpp, and so on, this should be an easy process. If not, cd to `src/include` directory, find the makefile_*machine-type*.inc.in that is closest to the machine you have, and start editting.

If all goes well, you should see the following executables built in the $INSTALLROOT/bin directory:

```
gridgen_model.exe
grib_prep.exe
hinterp.exe
vinterp.exe
siscan
staticpost.exe
```

All of the perl scripts for running the SI job are in $INSTALLROOT/etc:

```
window_domain_rt.pl
grib_prep.pl
wrfprep.pl
```

The install script also builds the EXT_DATAROOT, and DATAROOT directories depending on the environment variables set. Under EXT_DATAROOT, five subdirectories are created: `extprd, static, work, log` and GEOG (can ignored if GEOG_DATAROOT is defined). It will create DATAROOT directory.

When one is ready to run a different case, one will not need to reinstall SI. Instead, reset this environment variable:

```
setenv MOAD_DATAROOT your-new-case-directory
```

If you have chosen to build the GUI, the executable will be $INSTALLROOT/wrf_tools.

How to Run WRFSI?

The GUI is recommended to run the WRFSI program. For a detailed instruction on how to use the GUI, visit http://wrfsi.noaa.gov/. One of the advantages of using the GUI is that it has graphics to help you locate a domain. However, going through the following process may be helpful to understand how the various programs/scripts do and what they may produce. It may also be helpful in case you need to find out why something didn't quite work.

Here instructions are provided if you would like to run WRFSI manually.

Step 1: Localize the simulation domain and create static fields - *window_domain_rt.pl*

Set the environment variable for MOAD_DATAROOT, if it is different from the one you set when you install SI.

cd to $TEMPLATES/ directory, make a copy of the `default/` to your case directory:

```
cp -r default my-case
```

Then you need to remove file/directory protection for the directory and files in it:

```
chmod -R u+w my-case
```

cd to *my-case/*, and edit wrfsi.nl. If you have set GEOG_DATAROOT, EXT_DATAROOT, you'd find that these are incooperated in the wrfsi.nl file. The important namelist variables to edit at this time are those in `'hgridspec'` section as shown below:

```
&hgridspec
NUM_DOMAINS = 1
XDIM = 74
YDIM = 61
PARENT_ID = 1,
RATIO_TO_PARENT = 1,
DOMAIN_ORIGIN_LLI = 1,
DOMAIN_ORIGIN_LLJ = 1,
DOMAIN_ORIGIN_URI = 1,
DOMAIN_ORIGIN_URJ = 1,
MAP_PROJ_NAME = 'lambert',
MOAD_KNOWN_LAT = 34.726,
MOAD_KNOWN_LON = -81.226,
MOAD_STAND_LATS = 30.0, 60.0,
MOAD_STAND_LONS = -98.0
MOAD_DELTA_X = 30000.
MOAD_DELTA_Y = 30000.
SILAVWT_PARM_WRF = 0.
TOPTWVL_PARM_WRF = 2.
```

Once you have edited this portion of the namelist, you are ready to run. A typical run command looks like this:

```
setenv MOAD_DATAROOT $DATAROOT/my_case

$INSTALLROOT/etc/window_domain_rt.pl -w wrfsi -t $TEMPLATES/my-case
```

Other options are available for this perl script. Type the following to see them all:

```
$INSTALLROOT/etc/window_domain_rt.pl -h
```

The perl script makes use of the wrfsi.nl file you editted in the `$TEMPLATES/my-case/` directory to create netCDF control file and a working wrfsi.nl in `$MOAD_DATAROOT/static/` directory, and executes **gridgen_model.exe**. The perl script also creates directory siprd under `MOAD_DATAROOT` for running *wrfprep.pl* (see Step 3).

If it is successful, you should find a few directories created under `$MOAD_DATAROOT`: `static/, cdl/ and siprd/ and log/`. Check the `static/` directory and see if a file named '`static.wrfsi.d01`' is created. If so, you are done here. If not, check the `localization_domain.log.date` file in the `log/` directory, and try to identify errors.

The files in `cdl/` directory are the control files for netCDF output. The directory `siprd/` will be used to store SI output. Another file created by running this script is `wrfstatic_d01`, which is similar to the file '`static.wrfsi.d01`', but it conforms to WRF I/O API, and will be used when static-file-input option becomes available in WRF model for two-way and one-way nesting.

If you need to start over again, add option '`-c`' at the end of the above run command. It cleans the `MOAD_DATAROOT` directory:

```
$INSTALLROOT/etc/window_domain_rt.pl -w wrfsi -t $TEMPLATES/my-case -c
```

Step 2: Degrib GriB files - *grib_prep.pl*

It will be user's responsibility to find meteorological dataset to run WRF simulations. Once you have those files, place them in a unique directory for each case you work on. There are a number of sites you may be able to find data. See, for example, http://www.mmm.ucar.edu/wrf/users/downloads.html.

cd to `$EXT_DATAROOT/static/` directory, and edit *grib_prep.nl*. This is the namelist file for running **grib_prep.exe**. The critical ones to modify are the first rows of SRCNAME and SRCVTAB, and first line of SRCPATH:

```
SRCNAME = 'AWIP',
SRCVTAB = 'AWIP',
SRCPATH = '/public/data/40km_eta212_isobaric',
```

where SRCPATH is the directory name where your input GriB files are. A typical run command looks like this:

```
$INSTALLROOT/etc/grib_prep.pl -s 2000012412 -l 12 -t 6 AWIP
```

This run starts 2000012412, processes data for a 12 h forecast at 6 hour interval, and the data is from NAM/Eta in AWIP format which corresponds to Vtable.AWIP. As shown, the time information can be provided via the command line. If not, the values in the namelist will be used.

Hint: use the time interval between the available data only - there is practically no advantage to interpolate data to a time interval that is smaller than they are provided.

Other options on the comand line are available. Type the following to see them all:

```
grib_prep.pl -h
```

The perl script makes use of the namelist options in *grib_prep.nl*, and options on the command line to execute **grib_prep.exe**. The output from running this script should reside in `$EXT_DATAROOT/extprd/` directory, with file names beginning with AWIP:

```
AWIP:2004-06-30_00
AWIP:2004-06-30_06
AWIP:2004-06-30_12
```

If these files are not created, check `$EXT_DATAROOT/log/gp_AWIP.2004063000.log` to find clues.

Hint: **grib_prep.exe** runs the best when only the files relevant to this particular run are placed in one directory, defined in SRCPATH.

Step 3: Interpolating meterological data - *wrfprep.pl*

After you have successfully executed above two programs, you should be ready to do the final step in WRFSI: interpolating meteorological data to WRF grid. The *wrfprep.pl* script executes both ***hinterp*** and ***vinterp*** executables.

First, take another look at the namelist file, wrfsi.nl, and make sure everything is correctly specified in section `'interp_control'`:

```
&interp_control
NUM_ACTIVE_SUBNESTS = 0,
ACTIVE_SUBNESTS = 2,3,4,
PTOP_PA = 5000,
HINTERP_METHOD = 1,
LSM_HINTERP_METHOD = 1,
NUM_INIT_TIMES = 1,
INIT_ROOT = 'AWIP',
LBC_ROOT = 'AWIP',
LSM_ROOT = '',
CONSTANTS_FULL_NAME = '',
VERBOSE_LOG = .false.,
OUTPUT_COORD = 'ETAP',
LEVELS = 1.000 , 0.990 , 0.978 , 0.964 , 0.946 , 0.922 ,
0.894 , 0.860 , 0.817 , 0.766 , 0.707 , 0.644 ,
0.576 , 0.507 , 0.444 , 0.380 , 0.324 , 0.273 ,
0.228 , 0.188 , 0.152 , 0.121 , 0.093 , 0.069 ,
0.048 , 0.029 , 0.014 , 0.000
```

Make sure your INIT_ROOT, LBC_ROOT and/or LSM_ROOT are set to the correct data types you specified while running *grib_prep.pl*. These different root names allow one to use data from different sources.

To run the interpolation script, type the following:

```
$INSTALLROOT/etc/wrfprep.pl -s 2004063000 -f 12
```

Note the time information is provided at the command line. Again other options may be found by typing:

```
$INSTALLROOT/etc/wrfprep.pl -h
```

If successful, you should find the following files in $MOAD_DATAROOT/siprd/ directory:

```
hinterp.d01.2004-06-30_00:00:00
hinterp.d01.2004-06-30_06:00:00
hinterp.d01.2004-06-30_12:00:00
hinterp.global.metadata
wrf_real_input_em.d01.2004-06-30_00:00:00
wrf_real_input_em.d01.2004-06-30_06:00:00
wrf_real_input_em.d01.2004-06-30_12:00:00
```

The `wrf_real_input_em.d0.*` files are the ones to be used by WRF/real program, and these files are in netCDF format.

If you get here, your single domain WRFSI job should be successfully completed. If some files are not created, please check `$MOAD_DATAROOT/log/2004063000.wrfprep`, `2004063000.hinterp`, and `2004063000.vinterp` files for possible errors.

WRFSI GUI

For a detailed description and instruction on how to use it, please visit http://wrfsi.noaa.gov/.

If you have successfully installed the GUI, type the following to start it:

```
$INSTALLROOT/wrf_tools
```

Using WRFSI for Nesting

Running WRFSI for nesting option (data for more than one domain are created) is similar to running the program for a single domain. The key program for the nesting option is the ***gridgen_model*** program which, upon completion, will provide multiple static files, one for each domain specified. The static files are created for each domain independently. There is no consistency check between domains at this time, and it is up to the WRF model to make appropriate adjustment between the static fields when nesting is used.

There are a number of namelists that are key to set up a nested run. These are:

```
&hgridspec
NUM_DOMAINS = 2
XDIM = 74,
YDIM = 61,
PARENT_ID = 1, 1
RATIO_TO_PARENT = 1, 3
DOMAIN_ORIGIN_LLI = 1, 31
DOMAIN_ORIGIN_LLJ = 1, 17
```

```
DOMAIN_ORIGIN_URI = 74, 68
DOMAIN_ORIGIN_URJ = 61, 49
```

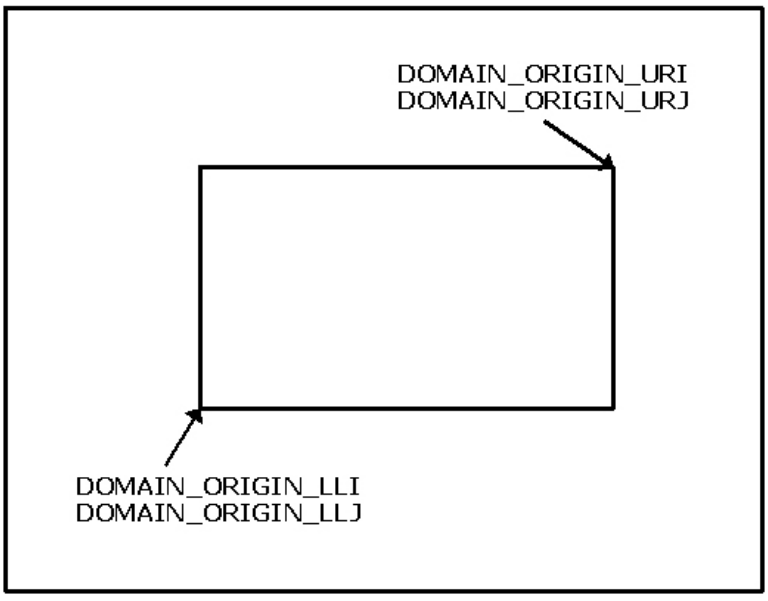
and

```
&interp_control
NUM_ACTIVE_SUBNESTS = 1,
ACTIVE_SUBNESTS = 2,
```

where, the ones under `&hgridspec` are used to create multiple `static.wrfsi.d0X` files; while the ones under `&interp_control` are used to create multiple SI output files for WRF model: `wrf_real_input_em.d01.*`, `wrf_real_input_em.d02.*`. The size of domain 2 in this case is (68 - 31) * 3 + 1 = 112, and (49 - 17) * 3 + 1 = 97.

`NUM_DOMAINS`: the number of domains one would like to create
`XDIM, YDIM`: here only the coarse domain dimensions are needed
`PARENT_ID`: a number identifying the domain
`RATIO_TO_PARENT`: integer, the ratio of coarse to fine domain grid distances
`DOMAIN_ORIGIN_LLI`: the starting nest location in terms of its parent domain index I
`DOMAIN_ORIGIN_LLJ`: the starting nest location in terms of its parent domain index J
`DOMAIN_ORIGIN_URI`: the ending nest location in terms of its parent domain index I
`DOMAIN_ORIGIN_URJ`: the ending nest location in terms of its parent domain index J (see figure below)

`NUM_ACTIVE_SUBNESTS`: the number of nests (excluding the parent domain)
`ACTIVE_SUBNESTS`: a list of the nests one would create SI output for



With these namelists set properly, a single run using window_domain_rt.pl, and wrfprep.pl will create all data required for a nested WRF run.

Similarly, for a two-nests, three domain total run, the above parameters should look like:

```
&hgridspec
NUM_DOMAINS = 3
XDIM = 74,
YDIM = 61,
PARENT_ID = 1, 1, 2
RATIO_TO_PARENT = 1, 3, 3
DOMAIN_ORIGIN_LLI = 1, 31, 41,
DOMAIN_ORIGIN_LLJ = 1, 17, 30,
DOMAIN_ORIGIN_URI = 74, 68, 72,
DOMAIN_ORIGIN_URJ = 61, 49, 60,
```

and

```
&interp_control
NUM_ACTIVE_SUBNESTS = 2,
ACTIVE_SUBNESTS = 2,3,
```

Again the GUI is recommended here. The design of the nest domains can be greatly simplified with the GUI.

Using Multiple Data Sources

Sometimes one would like to use combined data sources. For example, one may like to use sea-surface temperature (SST) from a different data source than the other meteorological fields, or one may like to use fields for land-surface model from a different data source. SI does support this function. To do so, one needs to run *grib_prep.pl* twice: Once for the special fields (such as SST, or LSM fields), and a second time for the rest of fields. Since these special fields are typically only needed for the model initial time, one needs only to process it for one time period. Hence the command is:

```
$INSTALLROOT/etc/grib_prep.pl -s 2004063000 -l 0 SST
```

Here the source for SST data is generically declared as SST, which would require the presence of Vtable.SST. Similarly, if the LSM fields come from another source, such as AGRMET, one would issue this command:

```
$INSTALLROOT/etc/grib_prep.pl -s 2004063000 -l 0 AGRMET
```

When one uses a different source to get the LSM fields, it is probably a good idea to remove these fields from the Vtable one uses to obtain all other one. After obtaining the degribbed files (these files would be named SST:[date_string], and AGRMET:[date_string]), one needs to edit *wrfsi.nl* so the wrfprep.pl knows how to use these multiple sourced input.

There are two ways different source data are used. One way is through the use of namelist variable 'CONSTANT_FULL_NAME'. This usually works with a single field like SST. In this case, edit wrfsi.nl so that 'CONSTANT_FULL_NAME' is set to:

```
CONSTANT_FULL_NAME = 'SSTDATA',
```

The perl script will link the SST file you produced to SSTDATA.

If you would like to use a different data source for LSM, then edit *wrfsi.nl* so that these variables are set to the following:

```
INIT_ROOT = 'GFS',
LBC_ROOT = 'GFS',
LSM_ROOT = 'AGRMET',
```

A special example to use multiple data source is the NCEP/NCAR Reanalysis Project (NNRP) data. Even though the data source is one, but because of the way data are archived, the upperair and surface data have different data dimensions. In order to use both surface and upperair data, one needs to run grib_prep.pl twice: once for the surface data using Vtable.NNRPSFC, and once for the upperair data using Vtable.NNRP (both provided). After that set the following in wrfsi.nl:

```
INIT_ROOT = 'NNRP',
LBC_ROOT = 'NNRP',
LSM_ROOT = 'NNRPSFC',
```

The latest year of NNRP data can be downloaded from WRF Users' page under the link DOWNLOAD.

Checking WRFSI Output

There are several ways you may check the output from SI. If you can find all the files that are supposed be created, then chances are good that everything has gone well. However if you need some sanity check, here are a few ways to go about it. These includes graphics tools, and simply print outs.

Checking output from *grib_prep*

Use the utility program, plotfmt.exe, described above to make plots of the degribbed files. This will be useful especially if you are ingesting data from a source other than those supported by the program.

Checking output from *gridgen_model*

There are three ways you may check the static fields created by *gridgen_model*. The first way is to use the netCDF utility, nudump, to check the output. You may type

```
ncdump -h static.wrfsi.d01
```

to get an idea (or header information) on what fields are in this file. You may also type

```
ncdump -v variable-name static.wrfsi.d01
```

to see the actual values for the variable you are interested.

The second way to check the output is to use the `read_wrf_nc.f` (see Chapter 8 for more information). Options provided by this program will allow you to check the data in various ways.

The third way to check the *gridgen_model* output is to use the NCL script provided by the WRFSI tar file. If you have used the GUI, you may be able to view the plots already. But if you want to create the plots outside the GUI, you can too. This can be done by cd `graphics/ncl` directory, link the *static.wrfsi.d01* file to *static.cdf* file, and run any of the ncl scripts residing in the directory. For example, if you would like to see a plot of terrain, use *avc.ncl* (and type '`ncl < avc.ncl`', or '`ncl avc.nl`' in the latest NCL).

Checking output from *hinterp*

The utility program to use here is the siscan program described above.

Checking output from *vinterp*

By this stage, the output files are in WRF I/O API-conforming netCDF format, so various supported post-processing tools may be used. See again Chapter 8 for more information. The program `read_wrf_nc.f` can also be used.

Description of the Namelist Variables

A. PROJECT_ID Section

This section is used to fill in metadata entries that document the run. The settings in this section will not affect the WRFSI run, but are provided for convenience to allow the user to document their run in the output metadata.

1. `SIMULATION_NAME`
Set this to a string describing the experiment or run.

2. `USER_DESC`
Set this to a string describing who is running the configuration.

B. FILETIMESEPC Section

This section provides the start and stop times that bound the period for which you want SI to produce data. Times are in UTC.

1. START_YEAR: 4-digit UTC year for start time.
2. START_MONTH: 2-digit UTC month for start time.
3. START_DAY: 2-digit UTC day of month for start time.
4. START_MINUTE: 2-digit UTC minute of month for start time.
5. START_SECOND: 2-digit UTC second of month for start time.
6. END_YEAR-END_SECOND: Same as 1-5, but for ending time.
7. INTERVAL: Interval in seconds between output times. No need to specify data interval to be finer than the available data times.

Note, the starting and ending times, and data interval may be overwritten on the command line when running wrfprep.pl.

C. HGRIDSPEC section

This is the section used to define your horizontal WRF grid.

1. NUM_DOMAINS: Integer number of nests, including the parent domain. If you are setting up for a single domain run, set to 1.

2. XDIM/YDIM: Integer number of points in the west-east and south-north directions, respectively. There are NUM_DOMAINS entries required. The first entry refers to the main grid (Mother Of All Domains or MOAD). Until nesting is supported, only the first will be used.

NOTE: The WRFSI defines the map to be what we refer to as the "non-staggered" grid. This implementation of the Arakawa-C stagger assumes all 3 component grids (U, V, and mass) are staggered with respect to these points. The U grid is staggered 0.5 gridpoints up w.r.t. the defined non-staggered points, the V grid is staggered 0.5 gridpoints to the right of the non-staggered points, and the mass grid is staggered 0.5 grid points up and 0.5 grid points to the right. To illustrate, here is an example for a case where (XDIM,YDIM) = (4,4):

```
+ V + V + V +
U T U T U T U
+ V + V + V +
U T U T U T U
+ V + V + V +
U T U T U T U
+ V + V + V +
```

The (+) points are the points exactly defined by the parameters in this namelist. The (T) points are the points on which the mass variables will be provided to and output by the WRF forecast model. The (U) points are the points on which the U momentum variables will be provided to and output by the WRF model. The (V) points are the points on which the V momentum variables will be provided to and output by the WRF model. Thus, if

your WRFSI configuration uses dimensions (XDIM, YDIM), the model will output the following:

Mass variables with dimensions (XDIM-1,YDIM-1)
U-momentum with dimensions (XDIM,YDIM-1)
V-momentum with dimensions (XDIM-1,YDIM)

3. PARENT_ID: Integer that represents the number of this nests parent nest. Note that the MOAD has no parent, and thus the first entry of PARENT_ID is always set to 1.

4. RATIO_TO_PARENT: Integer specifying the nest ratio of each nest to its parent in terms of grid spacing.

5. DOMAIN_ORIGIN_LLI/DOMAIN_ORIGIN_LLJ: This parameter specifies the (i,j) location of the nest grid's origin in its parent.

6. DOMAIN_ORIGIN_URI/DOMAIN_ORIGIN_URJ: This parameter specifies the (i,j) location of the nest grid's ending points in its parent.

7. MAP_PROJ_NAME: Character string specifying type of map projection. Valid entries are:

"polar" -> Polar stereographic
"lambert" -> Lambert conformal (secant and tangent)
"mercator" -> Mercator

8. MOAD_KNOWN_LAT/MOAD_KNOWN_LON: Real latitude and longitude of the center point in the grid. Values are in degrees, with positive latitude for the northern hemisphere and negative latitude for western hemisphere. Latitude must be between -90 and 90, and longitude between -180 and 180.

9. MOAD_STAND_LATS: 2 real values for the "true" latitudes (where grid spacing is exact). Must be between -90 and 90, and the values selected depend on projection:

Polar-stereographic: First value must be the latitude at which the grid spacing is true. Most users will set this equal to their center latitude. Second value must be +/-90. for NH/SH grids.

Lambert Conformal: Both values should have the same sign as the center latitude. For a tangential lambert conformal, set both to the same value (often equal to the center latitude). For a secant Lambert Conformal, they may be set to different values.

Mercator: The first value should be set to the latitude you wish your grid spacing to be true (often your center latitude). Second value is not used.

10. MOAD_STAND_LONS: This is one entry specifying the longitude in degrees East (-180->180) that is parallel to the y-axis of your grid, (sometimes referred to as the orientation of the grid). This should be set equal to the center longitude in most cases.

11. MOAD_DELTA_X/MOAD_DELTA_Y: Floating point values specifying grid spacing in meters in the west-east and north-south directions, respectively. For now, these two values must be the same.

12. SILAVWT_PARM_WRF: valid values are 1, 2 and 3, which give varying smoothness of the terrain field. The value of 3 gives the steepest terrain representation.

13. TOPTWVL_PARM_WRF: also controls smoothness of the terrain. The smaller the number, the rougher the terrain is.

D. SFCFILES Section

This section is used to specify the paths to the tiled global geographical data sets, which are obtained from ftp://aftp.fsl.noaa.gov/divisions/frd-laps/WRFSI/Geog_Data
IT IS NECESSARY THAT EACH DATA SET BE IN ITS OWN SUBDIRECTORY!

1. TOPO_30S: Path to the USGS-derived 30-second topographical height data.

2. LANDUSE_30S: Path to the tiled 24-category USGS 30-second land usage categorical data.

3. SOILTYPE_TOP_30S: Path to the FAO top-layer 16-category soil-type data.

4. SOILTYPE_BOT_30S: Same as (4) but for bottom layer.

5. GREENFRAC: Path to the greenness fraction data. Resolution: 0.15 degree.

6. SOILTEMP_1DEG: Path to the annual mean deep-layer temperature data. Resolution: 1 degree.

7. ALBEDO_NCEP: Path to the monthly climatological albedo data set (normalized to local zenith). Resolution: 0.15 degree.

8. MAXSNOWALB: Path to climatological maximum snow albedo data. Resolution: 0.15 degree.

9. ISLOPE: slope data. (Not yet used by WRF). Resolution: 1 degree.

E. INTERP_CONTROL section

This section controls the horizontal and vertical interpolation of the input gridded data sets.

1. NUM_ACTIVE_SUBNESTS: integer number of nests, excluding the parent domain.

2. ACTIVE_SUBNESTS: a list of the nests to create SI output for. For example, if you have set to configure 4 domains, then set this namelist to 2, 3, 4.

3. PTOP_PA: Specifies model top in Pascals. Default is 5000 Pa.

4. HINTERP_METHOD: Integer specifying method of interpolation for atmospheric variables. Codes:

    0: Nearest neighbor (not recommended)
    1: 4-pt bilinear (use if input data has similar resolution as output)
    2: 16-point

5. LSM_HINTERP_METHOD: Integer specifying the method of interpolation used for the land-masked fields. Codes are same as above. Recommended default is 0 or 1. **NOTE**: FOR USERS WISHING TO USE THE BACKGROUND DATA'S LANDUSE AND SOIL TYPE CATEGORIES, THIS MUST BE SET TO 0 AND YOU MUST OBTAIN "VEGCAT" AND "SOILCAT" FROM YOUR INPUT DATA SET, WHERE VEGCAT IS A 2D ARRAY OF DOMINANT LANDUSE CATEGORY (USGS 24-CAT) AND SOILCAT IS THE 2D ARRAY OF FAO DOMINANT SOIL CATEGORY.

6. NUM_INIT_TIMES: Integer, currently set to 1. This controls the number of output times to use the prefix specified by "INIT_ROOT" and "LSM_ROOT". The idea is for future support of analysis "nudging". The code will use the data specified by the INIT_ROOT/LSM_ROOT for time periods 1:NUM_INIT_TIMES, then switch to "LBC_ROOT" for the remaining time periods. If set to 0, all data comes from LBC_ROOT and CONSTANTS_FULL_NAME. Most users will set this to 0. However, setting it to 1 allows the model to be initialize with a different source of data for the initial conditions and land surface than what is used for lateral boundary conditions.

7. INIT_ROOT: Prefix of data to use for 1:NUM_INIT_TIMES. The wrfprep.pl script will look in ANALPATH (see SI_PATHS section) for files with this prefix and a time string suffix valid for the desired time. This entry is only used if NUM_INIT_TIMES > 0.

8. LBC_ROOT: Prefix of data files to use for lateral boundary condition times. The wrfprep.pl script will link in all files in "LBCPATH" that have this prefix and a valid time suffix in the correct range.

9. LSM_ROOT: For each NUM_INIT_TIME (when NUM_INIT_TIMES >0), the wrfprep.pl script will link in a file with this prefix found in LSMPATH that matches in

time. This is designed to support data for the NOAH LSM coming from a source other than the INIT_ROOT.

10. CONSTANTS_FULL_NAME: Specifies a list of file names to look for in "CONSTANTS_PATH". Data contained in any of these files actually found will be used at every output time, and will take precedence over duplicate data found in LSM_ROOT/INIT_ROOT/LBC_ROOT files.

11. VERBOSE_LOG: Logical. Setting to true provides a lot of logging for troubleshooting purposes.

12. LEVELS: List of levels to use in the WRF model in ascending (atmospherically) order. These values range from 1.0 to 0.0.

F. SI_PATHS Section

Specify path to deGRIBed (grib_prep output) data files. In most cases all of these will be the same path ($EXT_DATAROOT/extprd).

1. ANALPATH: Path to search for files with INIT_ROOT prefix when NUM_INIT_TIMES > 0.

2. LBCPATH: Path to search for files with LBC_ROOT prefix for all time periods > NUM_INIT_TIMES.

3. LSMPATH: Path to search for files with LSM_ROOT prefix for all time periods 0:NUM_INIT_TIMES.

4. CONSTANTS_PATH: Path to search for files with CONSTANTS_FULL_NAME for every time period.

List of Fields in WRFSI Output

A List of Fields

```
float ZNW(Time, bottom_top_stag) ;
float MU0(Time, south_north, west_east) ;
float T(Time, bottom_top, south_north, west_east) ;
float QVAPOR(Time, bottom_top, south_north, west_east) ;
float U(Time, bottom_top, south_north, west_east_stag) ;
float V(Time, bottom_top, south_north_stag, west_east) ;
float SPECHUMD(Time, bottom_top, south_north, west_east) ;
float PMSL(Time, south_north, west_east) ;
float SNOW(Time, south_north, west_east) ;
float TSK(Time, south_north, west_east) ;
float ST000010(Time, south_north, west_east) ;
float ST010040(Time, south_north, west_east) ;
float ST040100(Time, south_north, west_east) ;
```

```
  float ST100200(Time, south_north, west_east) ;
  float SM000010(Time, south_north, west_east) ;
  float SM010040(Time, south_north, west_east) ;
  float SM040100(Time, south_north, west_east) ;
  float SM100200(Time, south_north, west_east) ;
  float XICE(Time, south_north, west_east) ;
  float CANWAT(Time, south_north, west_east) ;
  float SOILHGT(Time, south_north, west_east) ;
  float XLAT(Time, south_north, west_east) ;
  float XLONG(Time, south_north, west_east) ;
  float LANDMASK(Time, south_north, west_east) ;
  float HGT(Time, south_north, west_east) ;
  float TOPOSTDV(Time, south_north, west_east) ;
  float TOPOSLPX(Time, south_north, west_east) ;
  float TOPOSLPY(Time, south_north, west_east) ;
  float COSALPHA(Time, south_north, west_east) ;
  float SINALPHA(Time, south_north, west_east) ;
  float F(Time, south_north, west_east) ;
  float E(Time, south_north, west_east) ;
  float MAPFAC_M(Time, south_north, west_east) ;
  float MAPFAC_U(Time, south_north, west_east_stag) ;
  float MAPFAC_V(Time, south_north_stag, west_east) ;
  float VEGFRA(Time, south_north, west_east) ;
  float SHDMAX(Time, south_north, west_east) ;
  float SHDMIN(Time, south_north, west_east) ;
  float ALBBCK(Time, south_north, west_east) ;
  float SNOALB(Time, south_north, west_east) ;
  float TMN(Time, south_north, west_east) ;
  float SLOPECAT(Time, south_north, west_east) ;
  float LANDUSEF(Time, land_cat, south_north, west_east) ;
  float SOILCTOP(Time, soil_cat, south_north, west_east) ;
  float SOILCBOT(Time, soil_cat, south_north, west_east) ;
```

Global Attributes in WRFSI Output File

```
        :corner_lats = 28.04805f, 44.23701f, 39.55859f, 24.53824f,
28.05492f, 44.24625f, 39.50478f, 24.49756f, 27.91458f, 44.37634f,
39.68672f, 24.41348f, 27.92145f, 44.38561f, 39.63279f, 24.37289f ;
        :corner_lons = -93.80435f, -92.59967f, -66.23782f, -72.82159f, -
93.95563f, -92.79419f, -66.07178f, -72.68448f, -93.81226f, -92.58652f,
-66.16806f, -72.86612f, -93.96326f, -92.78149f, -66.00174f, -72.72925f
;
        :WEST-EAST_GRID_DIMENSION = 74 ;
        :SOUTH-NORTH_GRID_DIMENSION = 61 ;
        :BOTTOM-TOP_GRID_DIMENSION = 28 ;
        :DX = 30000.f ;
        :DY = 30000.f ;
        :P_TOP = 5000.f ;
        :CEN_LAT = 34.83158f ;
        :CEN_LON = -81.02756f ;
        :FLAG_ST000010 = 1 ;
        :FLAG_ST010040 = 1 ;
        :FLAG_ST040100 = 1 ;
        :FLAG_ST100200 = 1 ;
```

```
:FLAG_SM000010 = 1 ;
:FLAG_SM010040 = 1 ;
:FLAG_SM040100 = 1 ;
:FLAG_SM100200 = 1 ;
:FLAG_TOPOSOIL = 1 ;
:simulation_name = "WRF Model Simulation" ;
:user_desc = "NCAR/MMM Test Case" ;
:si_version = 2 ;
:map_projection = "LAMBERT CONFORMAL" ;
:TITLE = "OUTPUT FROM WRF SI V02 PREPROCESSOR" ;
:START_DATE = "2000-01-24_12:00:00.0000" ;
:MOAD_CEN_LAT = 34.83158f ;
:STAND_LON = -98.f ;
:TRUELAT1 = 30.f ;
:TRUELAT2 = 60.f ;
:MAP_PROJ = 1 ;
:DYN_OPT = 2 ;
:ISWATER = 16 ;
:ISICE = 24 ;
:MMINLU = "USGS" ;
```

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2

Appendix B: Old Post-Processing Utilities

Table of Contents

- Introduction
- WRF2GrADS
- WRF2VIS5D

Introduction

The software packages WRF2GrADS and WRF2VIS5D are being phased out. The new package ARWpost will replace both of these packages.

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*). The ARW post-processing packages assume that the data from the ARW model is using the netCDF libraries.

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 GraDS and Vis5d:

- GrADS (*http://grads.iges.org/home.html*), requires the GrADS visualization software
- Vis5D (*http://www.ssec.wisc.edu/~billh/vis5d.html*), requires the Vis5D visualization software

WRF2GrADS

The WRF2GrADS converter reads ARW netCDF files, and creates "ieee", GrADS data files, and corresponding grads_control (.ctl) files.

The converter can process all ARW input, output and static (real and idealized data) in netCDF format.

Necessary software

- Obtain the WRF2GrADS TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
- GrADS software - You can download and install GrADS from http://grads/iges.org/grads

Hardware

The code has been ported to the following machines

- DEC Alpha
- Linux (pgf and intel compilers)
- MAC
- SUN
- SGI
- IBM

Steps to compile and run

Untar WRF2GrADS TAR file

Inside the TAR file you will have the following files:

Makefile README	
control_file control_file_height control_file_pressure	*Sample control file*
module_wrf_to_grads_netcdf.F module_wrf_to_grads_util.F wrf_to_grads.F	*Source code*
cbar.gs rgbset.gs	*Utility scripts*
skew.gs real_surf.gs plevels.gs	*Sample scripts to generate plots for real/idealized datasets*

rain.gs cross_z.gs zlevels.gs input.gs bwave.gs grav2d.gs hill2d.gs qss.gs sqx.gs sqy.gs	

Compile

- To compile the code, EDIT the **Makefile** to select the compiler flags for your machine
- Type: **make**
- This will create a **wrf_to_grads** executable

Edit the *control_file* file

```
-2
2000-01-24_12:00:00
2000-01-24_18:00:00
2000-01-25_00:00:00
end_of_time_list
```

Times to process

- If the first line contains a negative number, ALL times in the WRF file are processed.
- A positive number means process that number of times. In this case the times to process must be listed.
- The number in the first line, do not need to match the number of times listed. For the case where a positive number is used, the first *x* number of times will be processed.
- Do not remove or indent the line "end_of_time_list", the code depends on this line.

```
U             ! U Compoment of wind
V             ! V Component of wind
  UMET          ! U wind - rotated
  VMET          ! V wind - rotated
W             ! W Component of wind
THETA          ! Theta
TK            ! Temperature in K
TC            ! Temperature in C
TKE           ! TURBULENCE KINETIC
ENERGY
P             ! Pressure (HPa)
  Z             ! Height (m)
QVAPOR      ! Vapor
QCLOUD      ! Cloud Water
TSLB          ! SOIL TEMPERATURE
  SMOIS         ! SOIL MOISTURE
end_of_3dvar_list
```

3D variables to process

- List of all 3D variables you would like processed.
- If you do not wish to process a specific field, you can skip it, but simply indenting the line in which the field it listed. In this example, UMET, VMET, Z, and SMOIS will not be processed.
- If a variables is present in the WRF netCDF file, but not in this list, it can be processed by simply adding it to the list.
- To add a diagnostic, requires code changes.
- All 3D fields go here, including soil fields, which have a different number of levels.
- The "!" and description behind each field name is required by the program. If you add variables, remember to add the description of the field as well.

```
RAINC          ! TOTAL CUMULUS
PRECIPITATION
RAINNC         ! TOTAL GRID SCALE
PRECIPITATION
slvl            ! sea level pressure
T2             ! TEMP at 2 M
U10            ! U at 10 M
  U10M          ! U at 10 M - rotated
V10            ! V at 10 M
  V10M          ! V at 10 M - rotated
XLAT           ! LATITUDE
XLONG          ! LONGITUDE
XLAND          ! LAND MASK
end_of_2dvar_list
```

2D variables to process

- List of all 2D variables you would like processed.
- If you do not wish to process a specific field, you can skip it, but simply indenting the line in which the field it listed. In this case, U10M, and V10M will not be processed.
- If a variables is present in the WRF netCDF file, but not in this list, it can be processed by simply adding it to the list.
- To add a diagnostic, requires code changes.
- The "!" and description behind each field name is required by the program. If you add variables, remember to add the description of the field as well.
- Do not remove or indent the line "end_of_2dvar_list", the code depends on this line.

```
  /DATA/real/wrfinput_d01
wrfout_d01_2000-01-24_12:00:00
wrfout_d01_2000-01-25_00:00:00
  /DATA/b_wave/wrfout_d01
  /DATA/hill2d_x/wrfout_d01
end_of_file_list
```

WRF netCDF files process

- List of all the WRF netCDF files you would like processed.
- Do not mix different types of WRF files.
- If you do not wish to process a specific file, you can skip it, but simply indenting the line in which the field it listed. In this example, only the two real WRF output files will be processed.
- Multiple input file allowed, as long as they are listed in the correct order.
- At LEAST one input file is required.
- Do not remove or indent the line "end_of_file_list", the code it.

```
          ! what to do with the data
real      ! real / ideal / static
1         ! map background in grads
1         ! specify grads vertical grid
          ! 0=cartesian,
          ! -1=interp to z from lowest h
          ! 1 list levels (height/pressure)

1000.0
950.0
900.0
850.0
800.0
750.0
700.0
650.0
600.0
550.0
```

This section describes what to do with the data

- DO NOT ADD OR REMOVE LINES, the code needs this section exactly as is.
- We will process **real** data
- We would like a MAP background
- We would like to interpolate the data to levels given below

Levels to interpolate to

- This is only used if "1" is used for vertical interpolation above.
- Can use pressure (as in this example) or height levels.
- Levels must be from bottom to top.
- Pressure levels are given in mb, and height levels in km.
- Indenting will NOT remove a level from the list, it must be removed physically.

500.0
450.0
400.0
350.0
300.0
250.0
200.0
150.0
100.0

Run the code

- *wrf_to_grads control_file MyOutput [-options]*
  This will create **MyOutput.dat** and **MyOutput.ctl** for use with GrADS

- There are 3 debug levels (options) available:

  Only basic information will be written to the screen
  -v Debug option low
  -V Debug option high (lots of output)

- Now you are ready to use GrADS

Miscellaneous

To help users get started a number of GrADS scripts have been provided.

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 (Example, 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).

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

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/WRF2GrADS/GrADS.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 the Makefile.

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 the Makefile.

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.

WRF2VIS5D

Generate VIS5D files from WRF netCDF files.
ONLY ARW output files in netCDF format can be converted.

Necessary software

- Obtain the WRF2VIS5D TAR file from the WRF Download page (*http://www.mmm.ucar.edu/wrf/users/download/get_source.html*)
- VIS5D software (*http://www.ssec.wisc.edu/~billh/vis5d.html*)

Hardware

The code has been ported to the following machines

- DEC Alpha
- Linux
- SUN
- SGI
- IBM

Steps to compile and run

Untar WRF2VIS5D TAR file

Inside the TAR file, you will have the following files:

Makefile README	
wrf_v5d_input	*Sample control file*
module_map_utils.F module_wrf_to_vis5d_netcdf.F module_wrf_to_vis5d_util.F wrf_to_vis5d.F	*Source code*

Compile

- To compile the code, EDIT the **Makefile** to select the compiler flags for your machine
- Type: **make**
- This will create a **wrf_to_vis5d** executable

Edit *the control_file* file

```
-2
2000-01-24_12:00:00
2000-01-24_18:00:00
```

Times to process

- If the first line contains a negative number, ALL times in the WRF file are processed.
- A positive number means process that number of times. In this case the times to process must be listed.
- The number in the first line MUST match the number of times listed, for BOTH negative and positive numbers.

```
U
V
W
THETA
  TK
TC
QVAPOR
QCLOUD
QRAIN
RAINC
TSK
end_of_variable_list
```

Variables to process

- List of variables you would like process.
- If you do not wish to process a specific field, you can skip it, but simply indenting the line in which the field it listed. In this example, TK will not be processed.
- If a variables is present in the WRF netCDF file, but not in this list, it can be processed by simply adding it to the list.
- To add a diagnostic, requires code changes.
- Do not remove or indent the line "end_of_variable_list", the code depends on this line.

```
/real/wrfout_d01_2000-01-24_12:00:00
/real/wrfout_d01_2000-01-25_00:00:00
end_of_file_list
```

WRF netCDF files process

- List of all the WRF netCDF files you would like processed.
- List ONLY files you want to process. Indenting a file name will result in a run time error.
- Multiple input file allowed, as long as they are listed in the correct order.
- Do not remove or indent the line "end_of_file_list", the code depends on this line.

```
20     ! specify v5d vertical grid 0=cartesian, -
         1=interp to z from lowest h,
         >1 list levels (z) desired in vis5d file
```

This section describe what to do with the data

- 0 : cartesian vertical grid will be used
- -1 : interpolation from lowest h level
- >1 : for list of levels (height only, in km) to interpolate to (in this case 20 levels will be used)

```
1 1.
2 2.
3 3.
4 4.
5 5.
6 6.
7 7.
8 8.
9 9.
```

Levels to interpolate to

- Only height (must be in km).
- In this case, 20 levels must be given to correspond to number set above.
- Levels must be from bottom to top.
- Levels must be presided by the level number.
- Indenting will NOT remove a level from the list, it must be removed physically.

10 10.
11 11.
12 12.
13 13.
14 14.
15 15.
16 16.
17 17.
18 18.
19 19.
20 20.

Run the code

- wrf_to_vis5d wrf_v5d_input MyOutput
  This will create MyOutput for use with VIS5D

- Now you are ready to use VIS5D