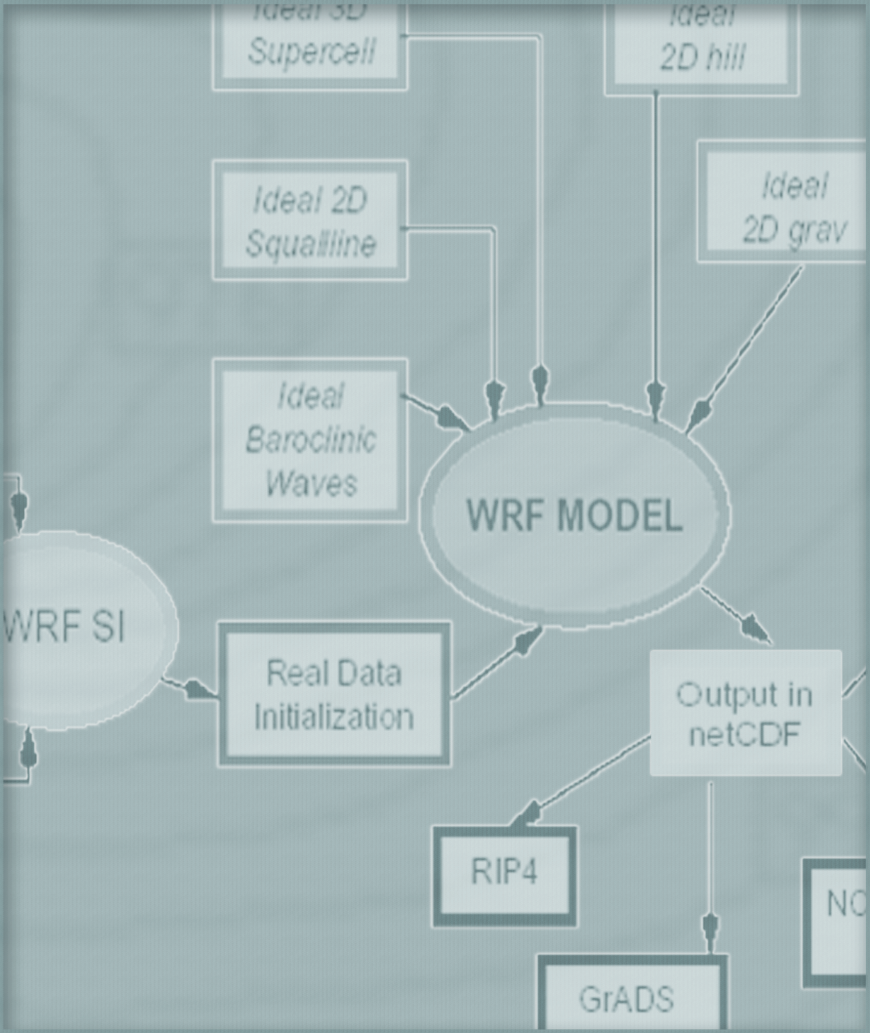
WEATHER RESEARCH & FORECASTING

ARW

Version 2 Modeling System User's Guide

July 2005



Mesoscale & Microscale Meteorology Division * National Center for Atmospheric Research

Foreword

This User's Guide describes the Advanced Research WRF (ARW) Version 2.1 modeling system, released in August 2005. As the ARW is developed further, this document will be continuously enhanced and updated. 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 feedbacks to wrfhelp@ucar.edu.

Contributors to this guide:
Wei Wang
Dale Barker
Cindy Bruyère
Jimy Dudhia
Dave Gill
John Michalakes

User's Guide for Advanced Research WRF (ARW) Modeling System Version 2.1

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

Chapter 1: Overview

Table of Contents

Introduction

The Advanced Research WRF (ARW) modeling system has been in development for the past few years. The current release is Version 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.
- Standard Initialization package (SI)
- 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

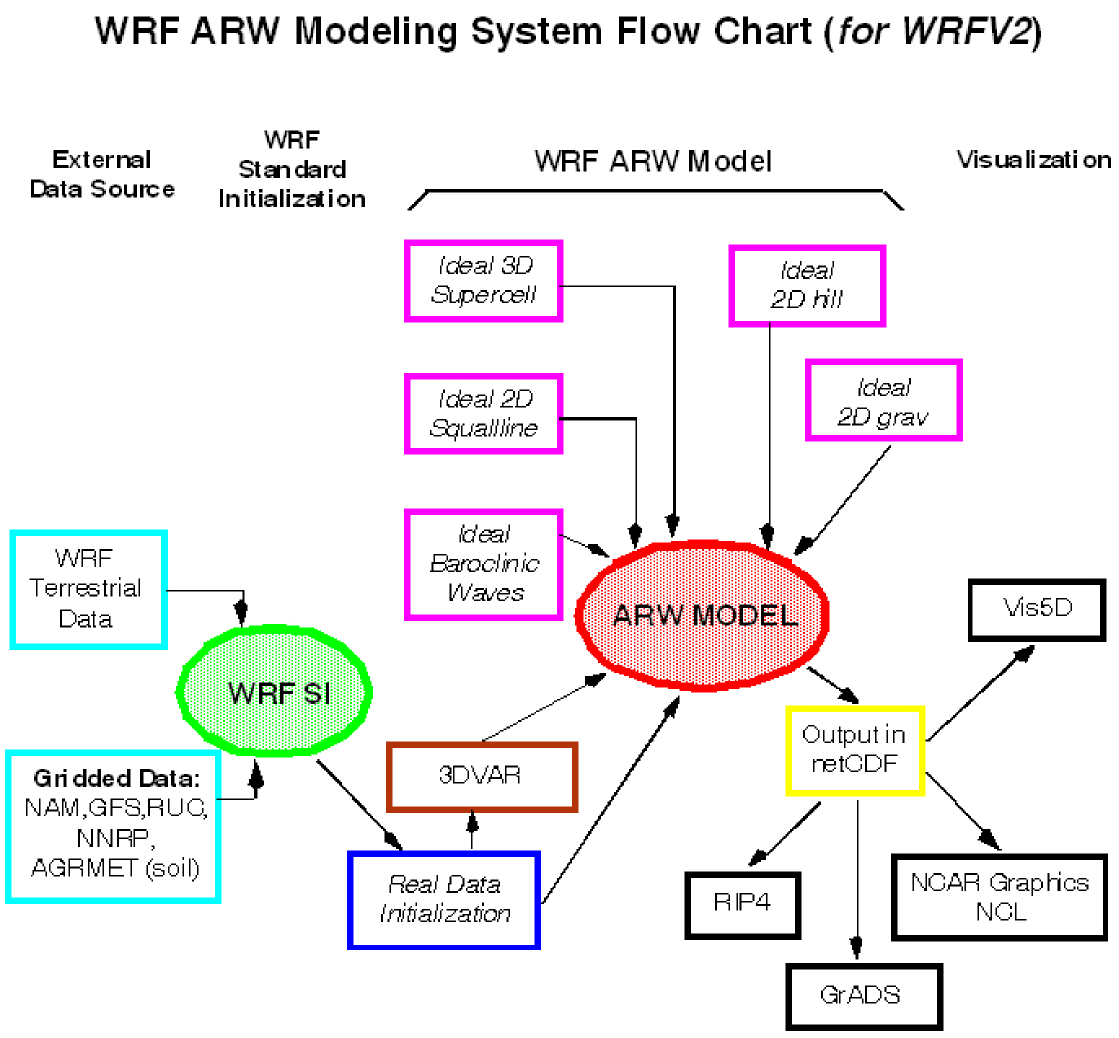
And these are the subjects of this document.

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:

- WRF Standard Initialization (WRFSI)
- WRF 3DVAR
- ARW solver
- Post-processing graphics tools

WRFSI

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 and to the model vertical coordinate.

WRF 3DVAR

This program is optional, but can be used to ingest observations into the interpolated analyses created by WRFSI. 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)
- 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

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

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 utilites 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
    - WRF v2 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 WRFV2.1.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 3DVAR Code

See details in Chapter 6.

Building the WRFSI Code

The simpliest build for SI code is to use all default directories. Several configure files are provided in ./wrfsi/src/include/ directory for various computers. A perl script, `install_wrfsi.pl` in the top directory is used to install the software.

- Get the latest WRFSI zipped tar file
    - wrfsi_v2.1.tar.gz
- unzip and untar the file
    - `gzip -cd wrfsi_v2.1.tar.gz | tar -xf -`
- `cd wrfsi`
- set the following environment variable to define where netCDF library and include directories are
    - `setenv NETCDF /path-to-netcdf`
- issue the following command to install - you may be prompted to answer whether you'd like to install the GUI:
    - `perl install_wrfsi.pl`
    - output from running the install script can be found in `make_install.log`
- `ls -l bin` (if environment variable INSTALLROOT is not set) or (if the environment variable INSTALLROOT is set, type) `ls -l $INSTALLROOT/bin`

More details can be found in Chapter 3.

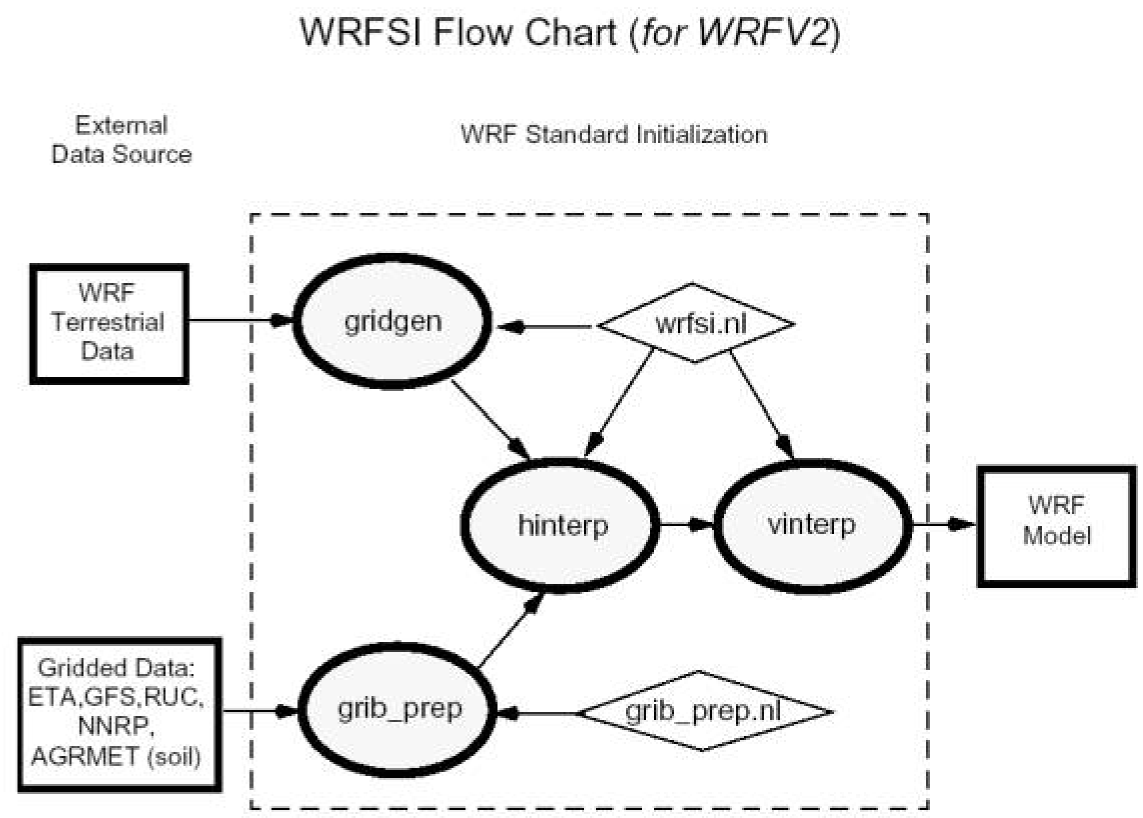
User’s Guide for Advanced Research WRF (ARW) Modeling System Version 2

Chapter 3: WRF Standard Initialization - Preparing Input Data

Table of Contents

Introduction

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:


WRFSI Flow Chart (for WRFV2)
External Data Source
WRF Standard Initialization
WRF Terrestrial Data
gridgen
wrfsi.nl
hinterp
vinterp
WRF Model
Gridded Data: ETA,GFS,RUC, NNRP, AGRMET (soil)
grib_prep
grib_prep.nl

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://wrfwi.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 ./wrfsi
setenv INSTALLROOT ./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
```

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

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 precess 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 $INSTALLROOT/domains/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 `$EXP_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, 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

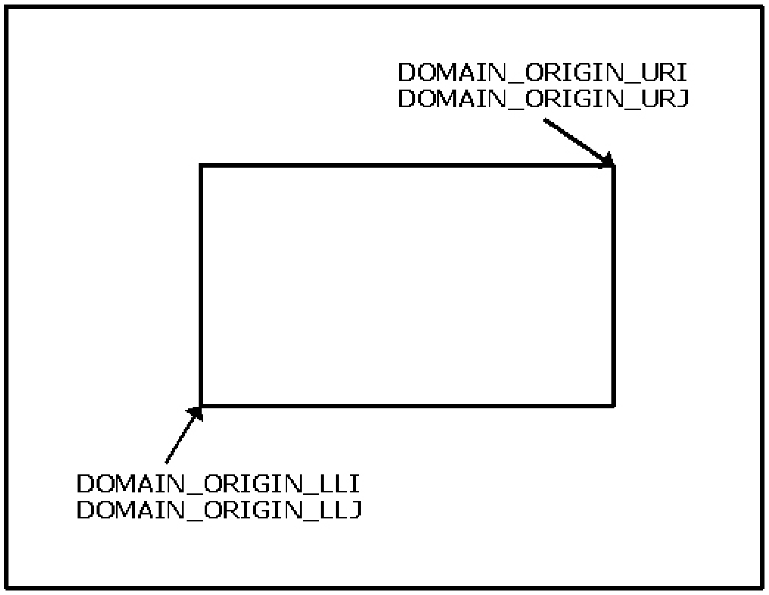
```
&interp_control
NUM_ACTIVE_SUBNESTS = 1,
ACTIVE_SUBNESTS = 2,
```

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
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 < anv.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

Chapter 4: WRF Initialization

Table of Contents

Introduction

The WRF model has two large classes of forecasts 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 Standard Initialization (SI)
- 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 Standard Initialization (SI) system. The data from the SI 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 SI 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 SI:

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

The convention is to use "wrf_real_input" to signify data that is output from the SI 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 SI output file has only a single time-slice of processed data.

The SI 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.
- The 3D data have already been vertically interpolated to the model's computational surfaces.
- 3D meteorological data from the SI: u, v, theta, mixing ratio
- 3D surface data from the SI: soil temperature, soil moisture, soil liquid
- 2D meteorological data from the SI: 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

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 SI files (the wrf_real* 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

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 2.1 supports a variety of capabilities. These include

- Real-data and idealized simulations
- Various lateral boundary condition options for both real-data and idealized simulations
- Full physics options
- non-hydrostatic and hydrostatic (runtime option)
- One-way, two-way nesting and moving nest

- 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 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. 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. If your path does not point to PGI/Intel-compiled netCDF, use NETCDF environment variable: e.g.,

```
setenv NETCDF /usr/local/netcdf-pgi
```

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

On NCAR's IBM computer (bluesky), 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 result in creating correct netCDF library and include link in *netcdf_links*/ directory under *WRFV2*/.

***Hint:* for nesting compile:**

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

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
`dyn_em/`	Directory for modules for dynamics in current WRF core (Advanced Research WRF core)
`dyn_exp/`	Directory for a 'toy' dynamic core
`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
`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***', 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. 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-pig

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.

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

Hint: if you are interested in making nested runs, make sure you select an option that supports nesting. In general, the options that are used for MPI/RSL/RSL_LITE are also 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**', 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). 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 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 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/** or **run/** directories. cd to those 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 : used for one-way nesting
wrf.exe : WRF model integration

Like in the idealized cases, these executables will be linked to **test/em_real** or **run/** directories. cd to one of these two directory to run the real-data case.

How to run WRF?

After 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, two-way nested, and one-way nested runs are explained on this page. Read on.

a. Idealized case

Say you choose to compile the test case **em_squall2d_x**, now type '***cd test/em_squall2d_x***' or '***cd run***'.

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 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 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**.

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).

NOTE: you must use the SI version 2.0 and above, to prepare input for V2 WRF!

Edit **namelist.input** for dates, and domain size. Also edit time step, output options, and physics options.

Type **'real.exe'** instead of ideal.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 IBM, the command is

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 interactive runs.

How to Make a 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), or multiple nest levels (telescoping). Moving nest option is also available since V2.0.3.1.

Most of options to start a nest run are handled through the namelist. ***All variables in the* namelist.input *file that have multiple columns of entries need to be edited with caution***. 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 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.

i_parent_start/j_parent_start: lower-left corner starting indices of the nest domain in its parent domain. 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 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 unit 2. This option allows the nest initialization to use interpolated 3-D meterological fields, 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, and use higher resolution input. Setting *fine_input_stream* = 0 is equivalent to the second option. This option is introduced in V2.1.

The way options 2 and 3 work is a bit cumbersome at this time. So please bear with us. It involves

- Running SI requesting one or more nest domains
- Running real.exe multiple times, once for each nest domain to produce *wrfinput_d0n* files (e.g. *wrfinput_d02*), and once for domain 1 like one

normally does to create *wrfbdy_d01* and *wrfinput_d01*
- Running WRF once

To prepare for the nested run, first follow the instruction in program WRF SI to create nest domain files. In addition to the files available for domain 1 (*wrf_real_input_em.d01.yyyy-mm-dd_hh:mm:ss* for all time periods), you should have a file from SI that is named *wrf_real_input_em.d02.yyyy-mm-dd_hh:mm:ss*, and this should be for the first time period of your model run. Say you have created SI output for a model run that starts at 1200 UTC Jan 24, 2000, using 6 hourly data, you should then have these files from SI:

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

If you use the nested option in SI, you should have one more file:

```
wrf_real_input_em.d02.2000-01-24_12:00:00
```

Once you have these files, do the following:

- Find out the dimensions of your nested domain. You can either make a note while you are running SI, or type

```
ncdump -h wrf_real_input_em.d02.2000-01-24_12:00:00
```

the grid dimensions can be found near the end of the dump: WEST-EAST_GRID_DIMENSION and SOUTH-NORTH_GRID_DIMENSION

- Use these dimension to set up the first namelist file. Note that you only need to set the namelist to run the first time period for domain 2. Only the first column matters in this case.
- To run **real.exe**, you must rename the domain 2 SI output file as if it is for domain 1. In the above case, type

```
mv wrf_real_input_em.d02.2000-01-
24_12:00:00 \
wrf_real_input_em.d01.2000-01-
24_12:00:00
```

Make sure you don't overwrite the real domain 1 input file for this time period. So do this first for the nest domain. Move or link file(s) for the nest to your run directory first.

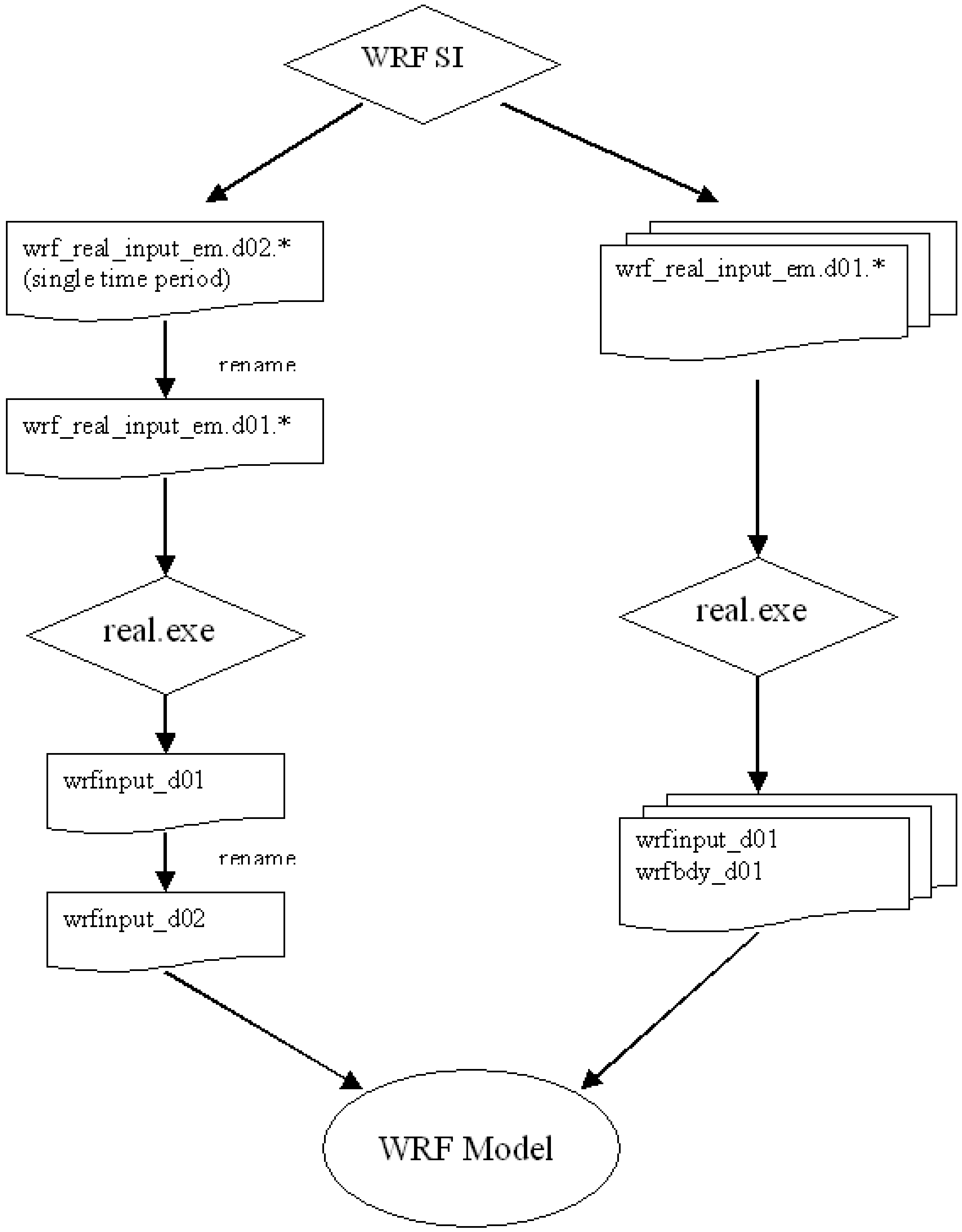
After running **real.exe**, rename `wrfinput_d01` to `wrfinput_d02`

Save the namelist edited for this domain.

- Re-edit the namelist for the coarse domain. Again, only the first column matters.
- Run **real.exe** for all domain 1 input files from SI, and create `wrfinput_d01` and `wrfbdy_d01` files.
- Edit the namelist again. This time you must pay attention to all the multiple column entries. Don't forget to set ***max_dom*** = the number of your nested domains, including the coarse domain.
- Run **wrf.exe** as usual with namelist modified

The following figure summarizes the data flow in the two-input, two-way nested run.

Data flow in the two-input, two-way nested run



How to Make a 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.)

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 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 an SI output for the nested domain (domain 2) named `wrf_real_input_em.d02.*`.

- Rename `wrf_real_input_em.d02.*` to `wrf_real_input_em.d01.*`. (Move the original domain 1 SI 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 as if it comes from SI again: `wrf_real_input_em.d02.*`. Make sure the date string is correctly specified in the file name.

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 `wrf_real_input_em.d02.*` file generated from Step 2 above. This will produce `wrfinput_d02` and `wrfbdy_d02` files.

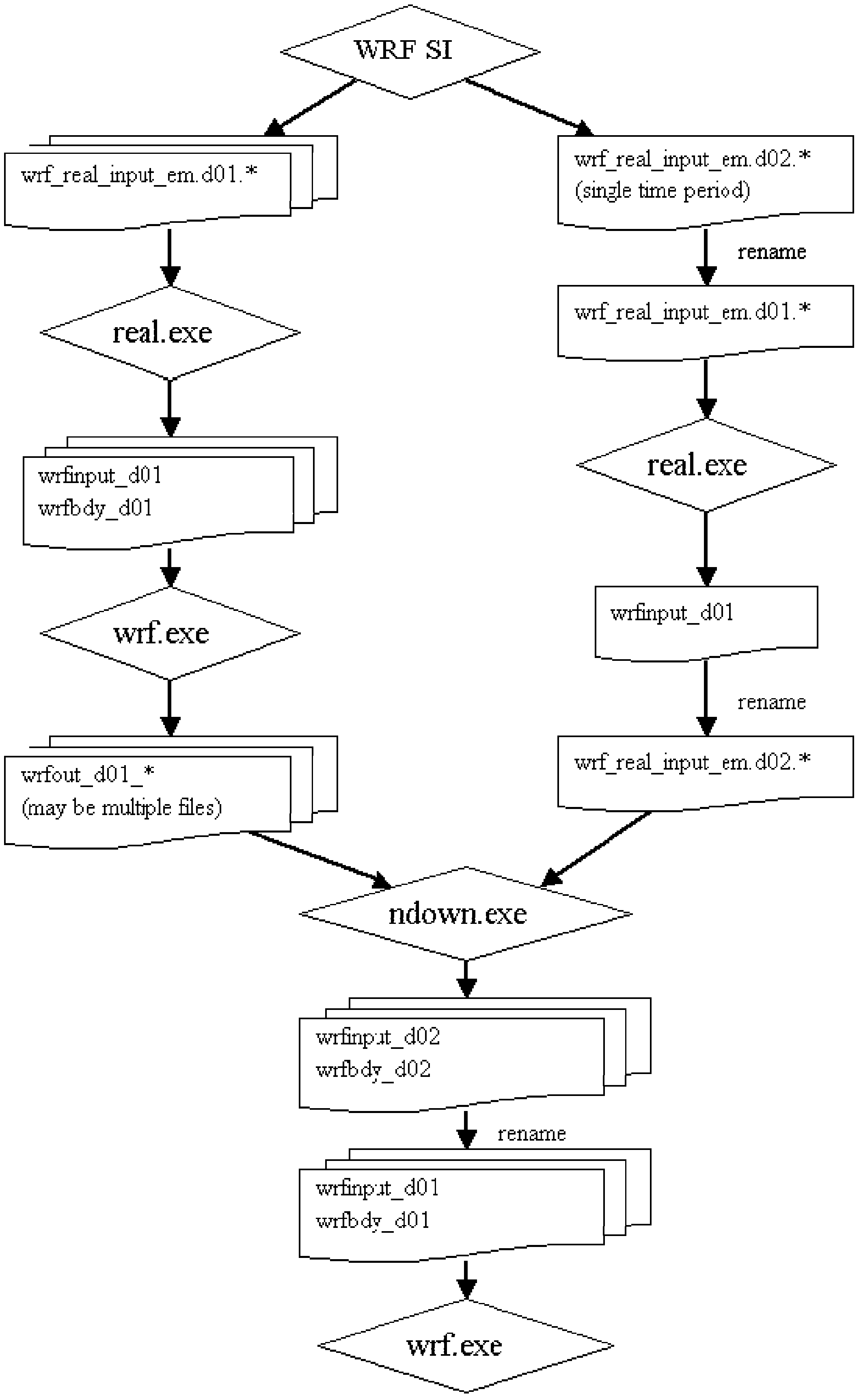
Note that one may run program ndown in mpi - if it is compiled so. For example,

```
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.


WRF SI
wrf_real_input_em.d01.*
wrf_real_input_em.d02.*
(single time period)
rename
wrf_real_input_em.d01.*
real.exe
real.exe
wrfinput_d01
wrfbdy_d01
wrfinput_d01
wrf.exe
rename
wrfout_d01_*
(may be multiple files)
wrf_real_input_em.d02.*
ndown.exe
wrfinput_d02
wrfbdy_d02
rename
wrfinput_d01
wrfbdy_d01
wrf.exe

How to Make a 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 needs to specify the nest movement in the namelist. The second option is to allow the nest to move automatically based on an automatic vortex-following algorithm. This option is designed to follow the movement of a tropical cyclone.

To make the specified moving nest runs, one first needs to compile the code with -DMOVE_NESTS 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. One must recompile with only –DMOVE_NESTS to do the specified moving nest runs.) 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.

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.

Both moving nest options are considered experimental at this time.

Physics and Diffusion 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 2.0.

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 related to Reisner2 with ice, snow and graupel processes suitable for high-resolution simulations.

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.

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.

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.

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.

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.

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.

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
`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
`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	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_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_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 real-data 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.
`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
&physics		Physics options
`chem_opt`	0	chemistry option - not yet available
`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	Thompson et al. 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
	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
	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)
`sf_sfclay_physics (max_dom)`		surface-layer option (old bl_sfclay_physics option)
	0	no surface-layer
	1	Monin-Obukhov scheme
	2	Monin-Obukhov (Janjic Eta) scheme
`sf_surface_physics (max_dom)`		land-surface option (old bl_surface_physics option)
	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
`surface_input_source`	1,2	where landuse and soil category data come from: 1 = SI/gridgen 2 = GRIB data from another model (only possible (VEGCAT/SOILCAT are in wrf_real_input_em files from SI)
`num_soil_layers`		number of soil layers in land surface model
	5	thermal diffusion scheme
	4	Noah landsurface model
	6	RUC landsurface model
`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
	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 auxinput_inname = "wrflowinp_d01", auxinput5_interval, and auxinput_end_h in namelist section *&time_control*
&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 when grid distance < 10 km)
	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 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 when grid distance < 10 km)
`damp_opt`		upper level damping flag (do not use for real-data cases until further notice)
	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)
`w_damping`		vertical velocity damping flag (for operational use)
	0	without damping
	1	with damping
`zdamp (max_dom)`	5000	damping depth (m) from model top
`dampcoef (max_dom)`	0.	damping coefficient (dampcoef <= 0.2, for 3D cases, set it <=0.1)
`base_temp`	290.	real-data, em ONLY, base sea-level temp (K)
`base_pres`	100000.	real-data, em ONLY, base sea-level pressure (Pa), DO NOT CHANGE
`base_lapse`	50.	real-data, em 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)
`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)
&bc_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
`miscelleneous in &domains:`		
`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

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

Introduction

To ensure the best communication of the WRF-VAR system, the documentation has been developed to be highly dynamic. For this reason it is best viewed online.

For a full description of the code and step-by-step help through a case study please visit:

http://www.mmm.ucar.edu/wrf/WG4/tutorial/wrf3dvar_tutorial.htm

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

Chapter 7: WRF Software

Table of Contents

Introduction

WRF Build Mechanism

The WRF build mechanism provides a uniform apparatus for configuring and compiling the WRF model and pre-processors over a range of platforms with a variety of options. This section describes the components and functioning of the build mechanism. For information on building the WRF code, see Section 2.

Required software:

The WRF build relies on Perl version 5 or later and a number of UNIX utilities: Csh and Bourne shell, make, M4, sed, awk, and the uname command. A C compiler is needed to compile programs and libraries in the tools and external directories. The WRF code itself is Fortran90. For distributed-memory, MPI and related tools and libraries should be installed.

Build Mechanism Components:

Directory structure: The directory structure of WRF consists of the top-level directory plus directories containing files related to the WRF software framework (frame), the WRF model (dyn_em, phys, share), configuration files (arch, Registry), helper programs (tools), and packages that are distributed with the WRF code (external).

Scripts: The top-level directory contains three user-executable scripts: configure, compile, and clean. The configure script relies on a Perl script in arch/Config.pl.

Programs: A significant number of WRF lines of code are automatically generated at compile time. The program that does this is tools/registry and it is distributed as source code with the WRF model.

Makefiles: The main makefile (input to the UNIX make utility) is in the top-level directory. There are also makefiles in most of the subdirectories that come with WRF. Make is called recursively over the directory structure. Make is not used directly to compile WRF; the compile script is provided for this purpose.

Configuration files: The configure.wrf contains compiler, linker, and other build settings, as well as rules and macro definitions used by the make utility. Configure.wrf is included by the Makefiles in most of the WRF source distribution (Makefiles in tools and external directories do not include configure.wrf). The configure.wrf file in the top-level directory is generated each time the configure script is invoked. It is also deleted by clean -a. Thus, configure.wrf is the place to make temporary changes: optimization levels, compiling with debugging, etc., but permanent changes should be made in arch/configure.defaults.

The arch/configure.defaults file contains lists of compiler options for all the supported platforms and configurations. Changes made to this file will be permanent. This file is used by the configure script to generate a temporary configure.wrf file in the top-level directory. The arch directory also contains the files preamble and postamble, which the unchanging parts of the configure.wrf file that is generated by the configure script.

The Registry directory contains files that control many compile-time aspects of the WRF code (described elsewhere). The files are named Registry.<em>core</em>. The configure script copies one of these to Registry/Registry, which is the file that tools/registry will use as input. The choice of <em>core</em> depends on settings to the configure script. Changes to Registry/Registry will be lost; permanent changes should be made to Registry.<em>core</em>.

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.

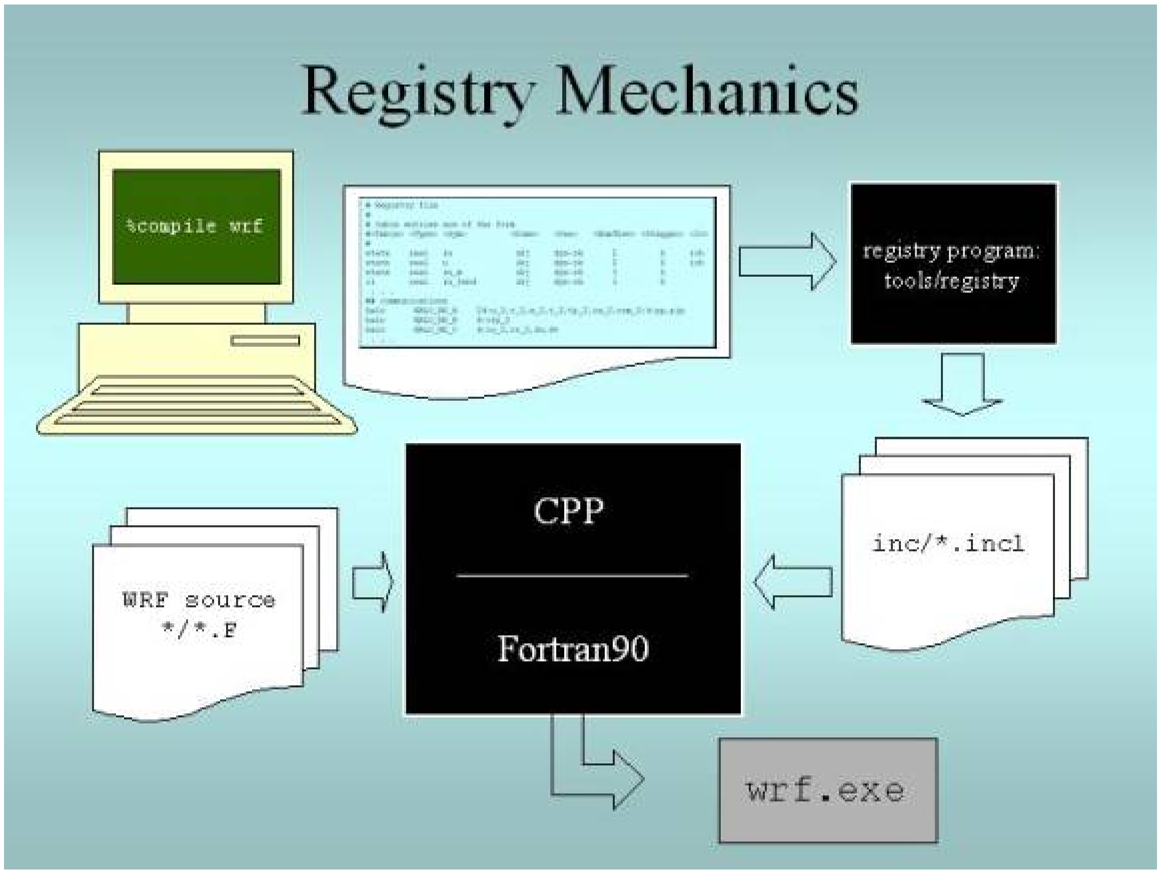


Figure 1. When the user compiles WRF, the Registry Program reads Registry/Registry, producing auto-generated sections of code that are stored in files in the inc directory. These are included into WRF using the CPP preprocessor and the Fortran compiler.

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

There are a number of visualization tools available to display ARW (*http://wrf-model.org/*) model data. Model data in netCDF format (netCDF libraries are available from the Unidata homepage (*http://www.unidata.ucar.edu/*) - registration login required), can essentially be displayed using any tool capable of displaying this data format. Currently 4 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 almost always required is the netCDF package from Unidata (*http://www.unidata.ucar.edu/* : login > Downloads > NetCDF). The ARW post-processing packages assume that the data from the ARW model is using the netCDF libraries.**

- **Additional libraries required by each of the 4 supported post-processing packages:**
  - NCL (*http://ngwww.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.

- **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://ngwww.ucar.edu/*)

- **Hardware**

  The code has been ported to the following machines
  - DEC Alpha
  - Linux
  - SUN
  - IBM

- **Steps to compile and run**

  - Untar WRF_NCL TAR file

    Inside the TAR file you will have the following files:

README_FIRST README_NCL gsn_code.ncl skewt_func.ncl wrf_plot.ncl wrf_user_mass.ncl	*Readme Files and* *NCL function scripts*
wrf_user_fortran_util_0.f make_ncl_fortran make_ncl_fortran.alpha make_ncl_fortran.linux make_ncl_fortran.sun	*FORTRAN utility file and* *make files to compile the utility program*
wrf_em_b_wave.ncl wrf_em_hill2d.ncl wrf_em_grav2d.ncl wrf_em_squall_2d_x.ncl wrf_em_squall_2d_y.ncl wrf_em_real_input.ncl wrf_em_real.ncl wrf_em_qc.ncl wrf_em_qss.ncl wrf_em_qv.ncl	*NCL scripts for ideal and real data*

wrf_em_sfc.ncl wrf_em_slp.ncl wrf_em_the.ncl	

- 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, running one of the **make_ncl_fortran** csh scripts that will build the shared-object library
    e.g. **make_ncl_fortran   wrf_user_fortran_util_0**
  - If successful, you will see these files created in your directory:
    so_locations
    wrf_user_fortran_util_0.o
    wrf_user_fortran_util_0.so
  - *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.

- Edit the script you want to run
  - Mostly it is only necessary to 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

- 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_em_real.ncl**
  - This will create a meta file
    e.g. wrf_mass_plots
  - The name of the meta file that is created, is controlled by the line:
    wks = wrf_open_ncgm("wrf_mass_plots")   inside the NCL script

- View the meta file
  - To view the meta file, use the command "idt", e.g.
    idt wrf_mass_plots
  - 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**
  - 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, 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.

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
  - 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 have 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*
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 will be created.

rip	Post-processing program. Before using this program, the input data must first be converted 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 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 of 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 be ended 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, testxxx.gif
  - A compete list of options are available inside the ncgm2gif script (*http://ngwww.ucar.edu/info/ncgm2gif*)

WRF2GrADS

The WRF2GrADS converter read 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 rain.gs cross_z.gs zlevels.gs input.gs bwave.gs grav2d.gs hill2d.gs qss.gs sqx.gs sqy.gs	*Sample scripts to generate plots for real/idealized datasets*

- 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 will be processed.
° A positive number means, process that number of times. In this case the times to process needs to 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 processing it, but simply indenting the line in which the field it listed. In this case, 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 for instance soil fields, which have a different number of levels.
° The "!" and description behind every 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_3dvar_list", the code depends on this line.

```
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
```

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 processing 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

```
XLAT            ! LATITUDE
XLONG          ! LONGITUDE
XLAND          ! LAND MASK
end_of_2dvar_list
```

simply adding it to the list.
° To add a diagnostic, requires code changes.
° The "!" and description behind every 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 processing it, but simply indenting the line in which the field it listed. In this case only the 2 real WRF output files will be processed.
° 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)
```

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

```
1000.0
950.0
900.0
850.0
800.0
750.0
700.0
650.0
600.0
550.0
500.0
450.0
400.0
350.0
300.0
250.0
200.0
150.0
100.0
```

Levels to interpolate to
° This is only used if "1" is used for vertical interpolation above.
° Can be pressure (as in this case) 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.

- 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 will be processed. ° A positive number means, process that number of times. In this case the times to process needs to 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 processing it, but simply indenting the line in which the field it listed. In this case, 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.

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

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.

- 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

read_wrf_nc utility

This utility was created to allow 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, 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 : [-help] [-head] [-m] [-M z] [-s] [-S x y z] ! [-t] [-v VAL]
    [-V VAL] [-w VAL]
    [-EditData]

Options:	*(Note: none of the options can be used with any other option)*
-help	Print help information.
-head	Print header information only.
-m	Print list of fields available for each time, plus the min and max values for each field. Also print the header information.
-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. Also print the header information.
-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. Also print the header information. **Default** if no options are supplied.
-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. Also print the header information.
-t	Print only the times in the file.
-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**

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 - **ALL times** for variable "VAR" WILL be changed.

- 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