# User's Guide for the NMM Core of the Weather Research and Forecast (WRF) Modeling System Version 2.1

# Chapter 6: WRF Software

## Table of Contents


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


## 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***, ***dyn_nmm***, ***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*** (for builds using the Eulerian Mass core, ARW) and ***Registry.NMM*** (for builds using the NMM core). The configure script copies one of these to ***Registry/Registry***, which is the file that tools/registry will use as input. The choice of ***Registry.EM*** or ***Registry.NMM*** depends on settings to the configure script. Changes to ***Registry/Registry*** will be lost; permanent changes should be made to ***Registry.EM*** or ***Registry.NMM*** depending on which core is used.

***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 the user's 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 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 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, em). For example to set it for WRF-NMM core, "***setenv WRF_NMM_CORE to 1"*** command should be issued. 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*** or compile ***nmm_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_nmm.exe*** (a preprocessor for real-data cases for the WRF-NMM) or for the WRF-ARW, ***real_em.exe*** (a preprocessor for real-data cases for the WRF-ARW) or ***ideal.exe*** (a preprocessor for idealized cases for the WRF-ARW), 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, ARW) 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 (DRAFT)" 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.

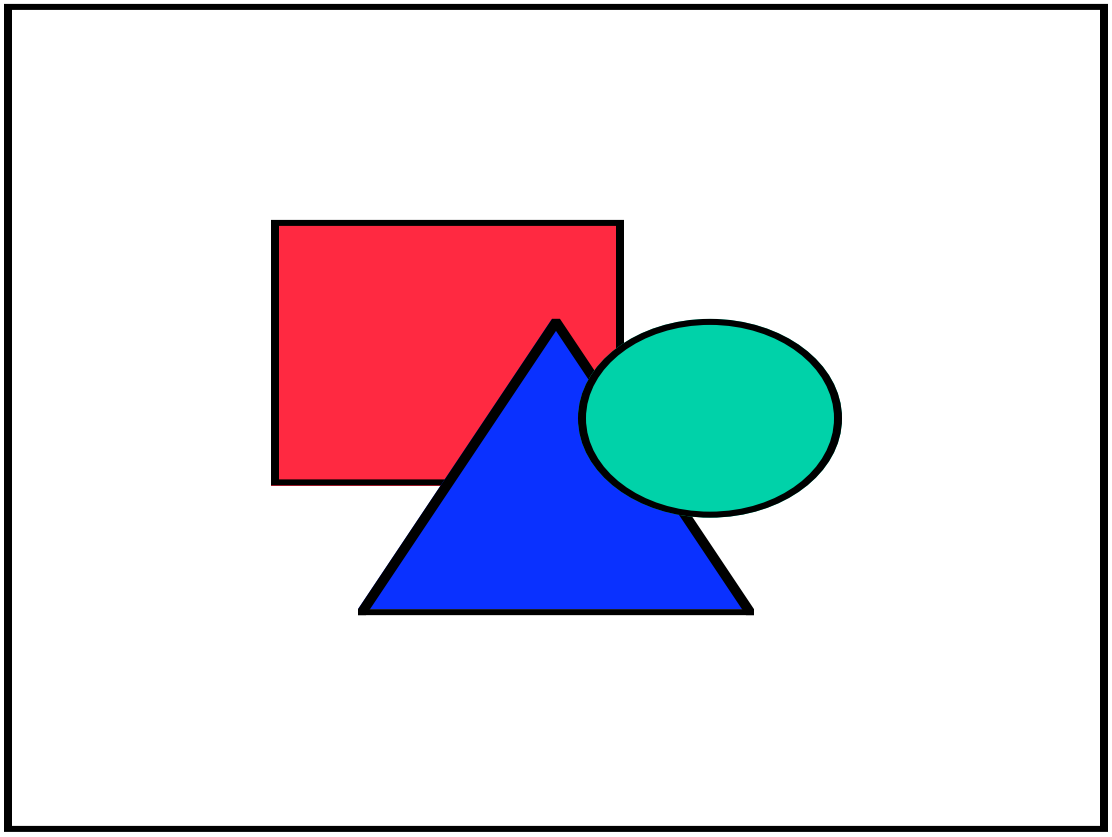

**Figure 1:** When the user compiles WRF, the Registry Program reads ***Registry/Registry***, producing auto-generated sections of code that are stored in files in the ***inc*** directory. These are included into WRF using the CPP preprocessor and the FORTRAN compiler.

In addition to the WRF model itself, the Registry/Registry file is used to build the accompanying preprocessors such as ***real_em.exe*** or ***real_nmm.exe*** (for real data) or ideal.exe (for WRF-ARW ideal simulations), and the ***ndown.exe*** program (used for the WRF-ARW 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 very high 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 developers is being developed by the WRF Training and Documentation Team, also known as the WRF Tiger Team (http://www.mmm.ucar.edu/wrf/WG2/Tigers).

Also, detailed subroutine-by-subroutine documentation has been implemented and is being maintained on-line. There are two web-based code browsing utilities available with WRF. One is a browser developed at the University of Oklahoma; the other is a code browser developed as part of the WRF project. These can be found on http://www.mmm.ucar.edu/wrf/WG2/software_2.0, along with short descriptions of the tools. The contents of these web pages are generated automatically from the WRF source code.

## Portability and Performance

WRF-ARW is supported on the following platforms:

| Vendor | Hardware | O.S. | Compiler |
|---|---|---|---|
| Cray Inc. | X1 | UNICOS | vendor |
| HP/Compaq | Alpha | Tru64 | vendor |
| | IA-64 (Intel) | Linux | Intel |
| | | HPUX | vendor |
| IBM | SP Power-*x* | AIX | vendor |
| SGI | IA-64 (Intel) | Linux | Intel |
| | MIPS | Irix | vendor |
| Sun | UltraSPARC | Solaris | vendor |
| various | IA-32/AMD 32 | Linux | Intel/PGI |
| various | IA-64/Opteron | Linux | Intel/PGI |

WRF-NMM is currently supported on the following platforms:

| Vendor | Hardware | O.S. | Compiler |
|---|---|---|---|
| IBM | SP Power-*x* | AIX | vendor |
| SGI | MIPS | IRIX | vendor |
| HP/COMPAQ/DEC | Alpha | Tru64 | vendor |
| Various | IA-32 | LINUX | PGI |
| Various | Opteron | LINUX | PGI |

Ports are in progress to other systems. Contact wrfhelp@ucar.edu for additional information.

For benchmark data, see http://www.mmm.ucar.edu/wrf/bench.