# WRF Registry and Examples

John Michalakes, NREL

Michael Duda, NCAR

Dave Gill, NCAR

WRF Software Architecture Working Group

## Outline

- Registry Mechanics

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

- Examples

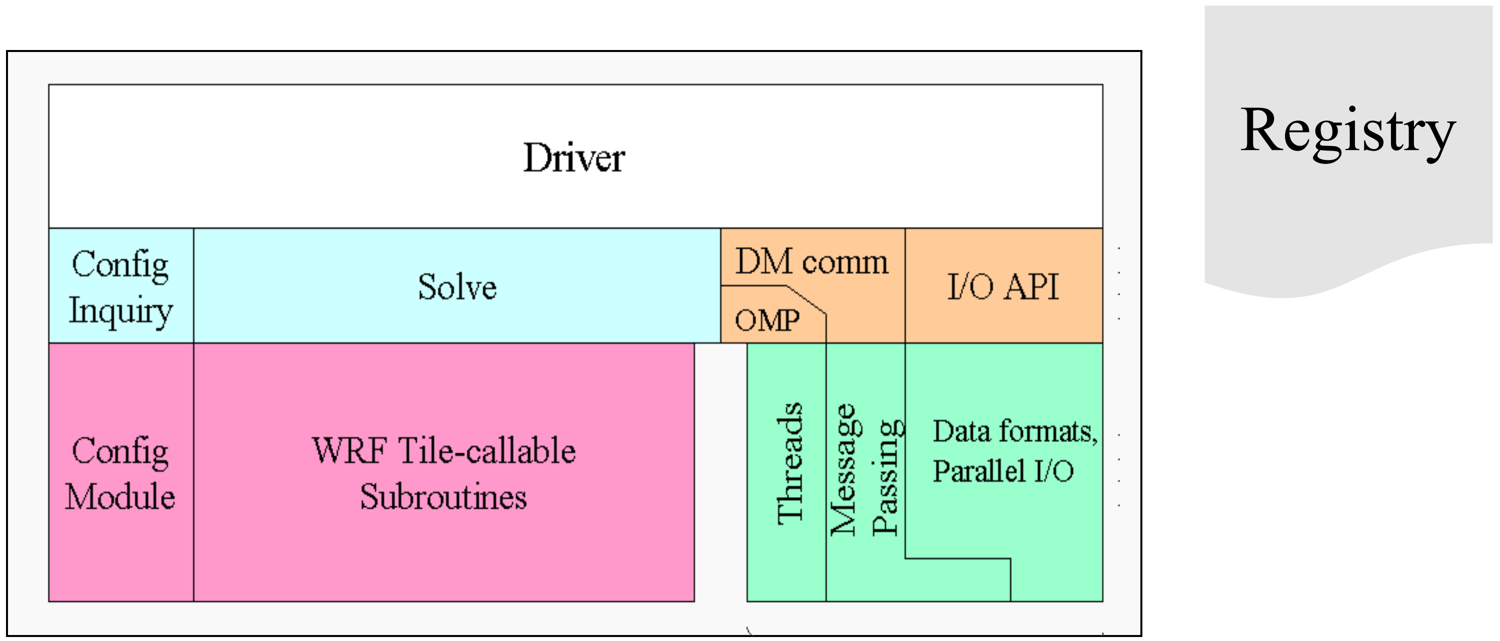
## Introduction – Intended Audience

- Intended audience for this tutorial session: scientific users and others who wish to:
  - Understand overall design concepts and motivations
  - Work with the code
  - Extend/modify the code to enable their work/research
  - Address problems as they arise
  - Adapt the code to take advantage of local computing resources

## Introduction – WRF Resources

- WRF project home page
  - http://www.wrf-model.org
- WRF users page (linked from above)
  - http://www.mmm.ucar.edu/wrf/users
- On line documentation (also from above)
  - http://www.mmm.ucar.edu/wrf/WG2/software_v2
- WRF user services and help desk
  - wrfhelp@ucar.edu

# WRF Software Architecture



- Hierarchical software architecture
  - Insulate scientists' code from parallelism and other architecture/implementation-specific details
  - Well-defined interfaces between layers, and external packages for communications, I/O, and model coupling facilitates code reuse and exploiting of community infrastructure, e.g. ESMF.

## WRF Registry

- "Active data-dictionary" for managing WRF data structures
  - Database describing attributes of model state, intermediate, and configuration data
    - Dimensionality, number of time levels, staggering
    - Association with physics
    - I/O classification (history, initial, restart, boundary)
    - Communication points and patterns
    - Configuration lists (e.g. namelists)
    - Nesting up- and down-scale interpolation

## WRF Registry

- "Active data-dictionary" for managing WRF data structures
  - Program for auto-generating sections of WRF from database:
    - 2000 - 3000 Registry entries ⇒ 300-thousand lines of automatically generated WRF code
    - Allocation statements for state data and I1 data
    - Interprocessor communications: Halo and periodic boundary updates, transposes
    - Code for defining and managing run-time configuration information
    - Code for forcing, feedback, shifting, and interpolation of nest data

## WRF Registry

- Why?
  - Automates time consuming, repetitive, error-prone programming
  - Insulates programmers and code from package dependencies
  - Allow rapid development
  - Documents the data
- A Registry file is available for each of the dynamical cores, plus special purpose packages
- Reference: Description of WRF Registry, http://www.mmm.ucar.edu/wrf/WG2/software_v2

## Registry Data Base

- Currently implemented as a text file: Registry/Registry.EM
- Types of entry:
  - *Dimspec* – Describes dimensions that are used to define arrays in the model
  - *State* – Describes state variables and arrays in the domain structure
  - *I1* – Describes local variables and arrays in solve
  - *Typedef* – Describes derived types that are subtypes of the domain structure

## Registry Data Base

- Types of entry:
  - *Rconfig* – Describes a configuration (e.g. namelist) variable or array
  - *Package* – Describes attributes of a package (e.g. physics)
  - *Halo* – Describes halo update interprocessor communications
  - *Period* – Describes communications for periodic boundary updates
  - *Xpose* – Describes communications for parallel matrix transposes
  - *Include* – Similar to a CPP #include file

## Registry State Entry

```
#       Type  Sym   Dims    Use      Tlev  Stag  IO          Dname      Descrip
state   real   u    ikjb    dyn_em    2     X     i01rhusdf   "U"     "X WIND COMPONENT"
```

- Elements
  - *Entry*: The keyword "state"
  - *Type*: The type of the state variable or array (real, double, integer, logical, character, or derived)
  - *Sym*: The symbolic name of the variable or array
  - *Dims*: A string denoting the dimensionality of the array or a hyphen (-)
  - *Use*: A string denoting association with a solver or 4D scalar array, or a hyphen
  - *NumTLev*: An integer indicating the number of time levels (for arrays) or hypen (for variables)

## Registry State Entry

```
#       Type  Sym   Dims    Use     Tlev  Stag  IO          Dname      Descrip

state   real   u    ikjb    dyn_em    2    X    i01rhusdf    "U"     "X WIND COMPONENT"
```

- Elements
  - *Stagger*: String indicating staggered dimensions of variable (X, Y, Z, or hyphen)
  - *IO*: String indicating whether and how the variable is subject to I/O and Nesting
  - *DName*: Metadata name for the variable
  - *Units*: Metadata units of the variable
  - *Descrip*: Metadata description of the variable

## Registry State Entry

```
#       Type  Sym   Dims     Use     Tlev  Stag      IO        Dname       Descrip

state   real   u    ikjb    dyn_em    2     X     i01rhusdf     "U"     "X WIND COMPONENT"
```

- This single entry results in over 100 lines of code automatically added to more than 40 different locations in the WRF model, the real and ideal initialization programs, and in the WRF-Var package

- Nesting code to interpolate, force, feedback, and smooth u
- Addition of u to the input, restart, history, and LBC I/O streams

## Registry State Entry

```
#      Type  Sym   Dims    Use     Tlev Stag    IO         Dname     Descrip

state  real   u    ikjb   dyn_em    2    X    i01rhusdf     "U"    "X WIND COMPONENT"
```

Declaration and dynamic allocation of arrays in TYPE(domain)

Two 3D state arrays corresponding to the 2 time levels of U

u_1 ( ims:ime , kms:kme , jms:jme )

u_2 ( ims:ime , kms:kme , jms:jme )

## Registry State Entry

```
#       Type  Sym  Dims    Use     Tlev Stag     IO        Dname      Descrip

state   real   u   ikjb   dyn_em    2    X    i01rhusdf    "U"    "X WIND COMPONENT"
```

Declaration and dynamic allocation of arrays in TYPE(domain)

Eight LBC arrays for boundary and boundary tendencies (dimension example for x BC)

u_b[xy][se] ( jms:jme, kms:kme, spec_bdy_width, 4 )

u_bt[xy][se] ( jms:jme, kms:kme, spec_bdy_width, 4 )

## State Entry: Defining a variable-set for an I/O stream

- Fields are added to a variable-set on an I/O stream in the Registry

```
#       Type  Sym  Dims   Use    Tlev  Stag    IO        Dname      Descrip
state   real  u    ikjb   dyn_em  2     X    i01rhusdf   "U"   "X WIND COMPONENT"
```

***IO*** is a string that specifies if the variable is to be subject to initial, restart, history, or boundary I/O. The string may consist of '**h**' (subject to history I/O), '**i**' (initial dataset), '**r**' (restart dataset), or 'b' (lateral boundary dataset). The 'h', 'r', and 'i' specifiers may appear in any order or combination.

## State Entry: Defining a variable-set for an I/O stream

- Fields are added to a variable-set on an I/O stream in the Registry

```
#       Type  Sym   Dims    Use     Tlev  Stag     IO       Dname     Descrip
state   real  u     ikjb    dyn_em  2     X     i01rhusdf   "U"   "X WIND COMPONENT"
```

The ‘h’ and ‘i’ specifiers may be followed by an optional integer string consisting of ‘0’, ‘1’, … , ‘9’ Zero denotes that the variable is part of the principal input or history I/O stream. The characters ‘1’ through ‘9’ denote one of the auxiliary input or history I/O streams.

**usdf** refers to nesting options: **u = UP, d = DOWN, s = SMOOTH, f = FORCE**

## State Entry: Defining Variable-set for an I/O stream

**`irh`** -- The state variable will be included in the WRF model input, restart, and history I/O streams

**`irh13`** -- The state variable has been added to the first and third auxiliary history output streams; it has been removed from the principal history output stream, because zero is not among the integers in the integer string that follows the character 'h’

## State Entry: Defining Variable-set for an I/O stream

**`rh01`** -- The state variable has been added to the first auxiliary history output stream; it is also retained in the principal history output

**`i205hr`** -- Now the state variable is included in the principal input stream as well as auxiliary inputs 2 and 5. Note that the order of the integers is unimportant. The variable is also in the principal history output stream

## State Entry: Defining Variable-set for an I/O stream

**`ir12h`** -- No effect; there is only 1 restart data stream

**`i01 --`** Data goes into real and into WRF

**`i1 --`** Data goes into real only

## Rconfig Entry

```
#          Type       Sym              How set               Nentries   Default
rconfig    integer spec_bdy_width  namelist,bdy_control        1           1
```

- This defines namelist entries
- Elements
  - *Entry*: the keyword “rconfig”
  - *Type*: the type of the namelist variable (integer, real, logical, string )
  - *Sym*: the name of the namelist variable or array
  - *How set*: indicates how the variable is set: e.g. namelist or derived, and if namelist, which block of the namelist it is set in

## Rconfig Entry

```
#          Type        Sym              How set             Nentries  Default
rconfig    integer spec_bdy_width  namelist,bdy_control     1          1
```

- This defines namelist entries
- Elements
  - *Nentries*: specifies the dimensionality of the namelist variable or array. If 1 (one) it is a variable and applies to all domains; otherwise specify max_domains (which is an integer parameter defined in module_driver_constants.F).
  - *Default*: the default value of the variable to be used if none is specified in the namelist; hyphen (-) for no default

## Rconfig Entry

```
#          Type         Sym                How set             Nentries   Default
rconfig    integer  spec_bdy_width  namelist,bdy_control          1           1
```

- Result of this Registry Entry:
  - Define an namelist variable "spec_bdy_width" in the bdy_control section of namelist.input
  - Type integer (others: real, logical, character)
  - If this is first entry in that section, define "bdy_control" as a new section in the namelist.input file
  - Specifies that bdy_control applies to all domains in the run

```
 --- File: namelist.input ---

&bdy_control
 spec_bdy_width             = 5,
 spec_zone                  = 1,
 relax_zone                 = 4,
    . . .
 /
```

## Rconfig Entry

```
#          Type       Sym                   How set              Nentries   Default
rconfig    integer  spec_bdy_width  namelist,bdy_control       1            1
```

- Result of this Registry Entry:
  - if Nentries is “max_domains” then the entry in the namelist.input file is a comma-separate list, each element of which applies to a separate domain
  - The single entry in the Registry file applies to each of the separate domains

```
 --- File: namelist.input ---

&bdy_control
 spec_bdy_width          = 5,
 spec_zone               = 1,
 relax_zone              = 4,
     . . .
 /
```

## Rconfig Entry

```
#           Type         Sym              How set               Nentries   Default
rconfig     integer spec_bdy_width  namelist,bdy_control       1           1
```

- Result of this Registry Entry:
  - Specify a default value of "1" if nothing is specified in the namelist.input file
  - In the case of a multi-process run, generate code to read in the bdy_control section of the namelist.input file on one process and broadcast the value to all other processes

```
 --- File: namelist.input ---

&bdy_control
 spec_bdy_width             = 5,
 spec_zone                  = 1,
 relax_zone                 = 4,
     . . .
 /
```

## Outline

- Registry Mechanics

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

- Examples
  - 0) Add output without recompiling
  - 1) Add a variable to the namelist
  - 2) Add an array
  - 3) Compute a diagnostic
  - 4) Add a physics package

## Example 0: Add output without recompiling

- Edit the namelist.input file, the time_control namelist record

```
iofields_filename = "myoutfields.txt” (MAXDOM)
io_form_auxhist7 = 2 (choose an available stream)
auxhist7_interval = 10 (MAXDOM, every 10 minutes)
```

- Place the fields that you want in the named text file **`myoutfields.txt`**

```
+:h:7:RAINC,RAINNC
```

- Where “**+**” means ADD this variable to the output stream, “**h**” is the history stream, and “**7**” is the stream number

## Example 0: Zap output without recompiling

- Edit the namelist.input file, the time_control namelist record

```
iofields_filename = "myoutfields.txt”
```

- Place the fields that you want in the named text file **`myoutfields.txt`**

```
-:h:0:W,PB,P
```

- Where “**-**” means REMOVE this variable from the output stream, “**h**” is the history stream, and “**0**” is the stream number (standard WRF history file)

## Example 1: Add a variable to the namelist

- Use the examples for the rconfig section of the Registry
- Find a namelist variable similar to what you want
  - Integer *vs* real *vs* logical *vs* character
  - Single value *vs* value per domain
  - Select appropriate namelist record
- Insert your mods in all appropriate Registry files

## Example 1: Add a variable to the namelist

- Remember that ALL Registry changes require that the WRF code be cleaned and rebuilt

```
./clean -a
./configure
./compile em_real
```

## Example 1: Add a variable to the namelist

- Adding a variable to the namelist requires the inclusion of a new line in the Registry file:

```
rconfig integer my_option_1  namelist,time_control  1 0 - "my_option_1" "test namelist option”
rconfig integer my_option_2  namelist,time_control  max_domains 0
```

- Accessing the variable is through an automatically generated function:

```
USE module_configure
INTEGER :: my_option_1 , my_option_2

CALL nl_get_my_option_1( 1, my_option_1 )
CALL nl_set_my_option_2( grid%id, my_option_2 )
```

## Example 1: Add a variable to the namelist

- You also have access to the namelist variables from the grid structure …

```
SUBROUTINE foo ( grid , ... )

   USE module_domain
   TYPE(domain) :: grid

   print *,grid%my_option_1
```

## Example 1: Add a variable to the namelist

- … and you also have access to the namelist variables from config_flags

```
SUBROUTINE foo2 ( config_flags , ... )

   USE module_configure
   TYPE(grid_config_rec_type) :: config_flags

   print *,config_flags%my_option_2
```

## Example 1: Add a variable to the namelist

- What your variable looks like in the namelist.input file

```
&time_control
run_days                                    = 0,
run_hours                                   = 0,
run_minutes                                 = 40,
run_seconds                                 = 0,
start_year                                  = 2006, 2006, 2006,
my_option_1                                 = 17
my_option_2                                 = 1, 2, 3
```

## Examples

- 1) Add a variable to the namelist
- 2) Add an array to solver, and IO stream
- 3) Compute a diagnostic
- 4) Add a physics package

## Example 2: Add an Array

- Adding a state array to the solver, requires adding a single line in the Registry
- Use the previous Registry instructions for a state or l1 variable

## Example 2: Add an Array

- Select a variable similar to one that you would like to add
  - 1d, 2d, or 3d
  - Staggered (X, Y, Z, or not "-", *do not leave blank*)
  - Associated with a package
  - Part of a 4d array
  - Input (012), output, restart
  - Nesting, lateral forcing, feedback

# Example 2: Add an Array

- Copy the "similar" field's line and make a few edits
- Remember, no Registry change takes effect until a "clean -a" and rebuild

```
state  real  h_diabatic  ikj  misc  1  -      r                          \
       "h_diabatic"  "PREVIOUS TIMESTEP CONDENSATIONAL HEATING"

state  real  msft        ij   misc  1  -     i012rhdu=(copy_fcnm)  \
       "MAPFAC_M"    "Map scale factor on mass grid"

state  real  ht          ij   misc  1  -     i012rhdus                  \
       "HGT"         "Terrain Height"

state  real  ht_input    ij   misc  1  -     -                          \
       "HGT_INPUT"   "Terrain Height from FG Input File"

state  real  TSK_SAVE    ij   misc  1  -     -                          \
       "TSK_SAVE"    "SURFACE SKIN TEMPERATURE"   "K"
```

## Example 2: Add an Array

- Always modify Registry.*core_name*, where *core_name* might be EM, for example

```
state   real   h_diabatic   ikj   misc   1   -      r                      \
        "h_diabatic"   "PREVIOUS TIMESTEP CONDENSATIONAL HEATING"

state   real   msft         ij    misc   1   -     i012rhdu=(copy_fcnm)  \
        "MAPFAC_M"     "Map scale factor on mass grid"

state   real   ht           ij    misc   1   -     i012rhdus             \
        "HGT"          "Terrain Height"

state   real   ht_input     ij    misc   1   -     -                     \
        "HGT_INPUT"    "Terrain Height from FG Input File"

state   real   TSK_SAVE     ij    misc   1   -     -                     \
        "TSK_SAVE"     "SURFACE SKIN TEMPERATURE"   "K"
```

## Example 2: Add an Array

- Add a new 3D array that is sum of all moisture species, called all_moist, in the Registry.EM
  - Type: real
  - Dimensions: 3D and ikj ordering, not staggered
  - Supposed to be output only: h
  - Name in netCDF file: all_moist

```
state    real   all_moist      ikj      \
dyn_em      1      -     h   \
"all_moist"  \
"sum of all of moisture species"     \
"kg kg-1"
```

## Example 2: Add an Array

- Registry state variables become part of the derived data structure usually called grid inside of the WRF model.
- WRF model top $\rightarrow$ integrate $\rightarrow$ solve_interface $\rightarrow$ solve
- Each step, the grid construct is carried along for the ride
- No source changes for new output variables required until below the solver routine

## Example 2: Add an Array

- Top of solve_em.F
- grid is passed in
- No need to declare any new variables, such as all_moist

```
!WRF:MEDIATION_LAYER:SOLVER

  SUBROUTINE solve_em ( grid , &

  config_flags , &
```

## Example 2: Add an Array

- In solve_em, add the new array to the call for the microphysics driver
- Syntax for variable=local_variable is a association convenience
- 0D, 1D, 2D, 3D state arrays are contained within grid, and must be de-referenced

```
CALL microphysics_driver(                &
     QV_CURR=moist(ims,kms,jms,P_QV),  &
     QC_CURR=moist(ims,kms,jms,P_QC),  &
     QR_CURR=moist(ims,kms,jms,P_QR),  &
     QI_CURR=moist(ims,kms,jms,P_QI),  &
     QS_CURR=moist(ims,kms,jms,P_QS),  &
     QG_CURR=moist(ims,kms,jms,P_QG),  &
     QH_CURR=moist(ims,kms,jms,P_QH),  &
     all_moist=grid%all_moist        ,  &
```

## Example 2: Add an Array

- After the array is re-referenced from grid and we are inside the microphysics_driver routine, we need to
  - Pass the variable through the argument list
  - Declare our passed in 3D array

```
   ,all_moist &




 REAL, DIMENSION(ims:ime ,kms:kme ,jms:jme ), &
       INTENT(OUT) ::  all_moist
```

## Example 2: Add an Array

- After the array is re-referenced from grid and we are inside the microphysics_driver routine, we need to
  - Zero out the array at each time step

```
!  Zero out moisture sum.

 DO j = jts,MIN(jde-1,jte)
 DO k = kts,kte
 DO i = its,MIN(ide-1,ite)
    all_moist(i,k,j) = 0.0
 END DO
 END DO
 END DO
```

## Example 2: Add an Array

- After the array is re-referenced from grid and we are inside the microphysics_driver routine, we need to
  - At the end of the routine, for each of the moist species that exists, add that component to all_moist

```
DO j = jts,MIN(jde-1,jte)
  DO k = kts,kte
  IF ( f_QV ) THEN
     DO i = its,MIN(ide-1,ite)
        all_moist(i,k,j) = all_moist(i,k,j) + &
                           qv_curr(i,k,j)
      END DO
  END IF
```