WRF Registry

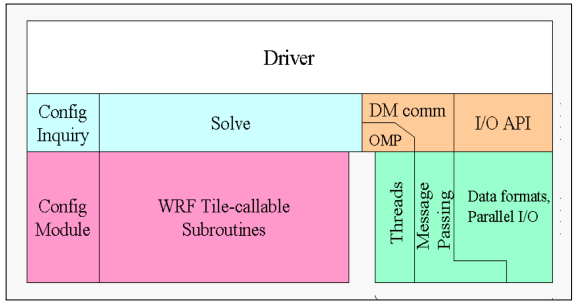
Wei Wang
July, 2018

Adapted from lectures by
John Michalakes, NRL, and Dave Gill, NCAR

Outline

- WRF Registry
  - What is WRF Registry?
  - Keyword syntax
  - Three commonly used ones
- Some Examples
  - Runtime I/O mods
  - Adding a variable to the namelist
  - Adding an array to WRF
  - Adding a passive tracer

WRF Software Architecture



Text based file for the model
Active data dictionary
Used to auto generate source
Controls/defines
Variables (I/O, comms, nesting)
Communications
namelist options

About 300k lines added to source
Easy – 3x the size since initial release
Compile-time option
./clean
./configure
./compile
Registry.EM_COMMON

What is a Registry?

- Currently implemented as a text file:
  - Registry/Registry.EM_COMMON for WRF ARW
- What does it do?
  - Defines model arrays
  - Used to auto-generate many thousand lines of code
  - Defines arrays for input and output
  - Defines how nest feedback is used
  - Defines namelists
  - Defines how arrays are used (memory management)
  - Defines MPI communications

Registry Keywords

- 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 Keywords

- 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  tsk   ij    misc    1    -     i01rhud     "TSK"     "SKIN TEMP"
```

- 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  tsk   ij    misc    1    -     i01rhud     "TSK"     "SKIN TEMP"
```

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

registry.dimspec: where dimensions are defined

```
#<Table>  <Dim> <Order> <How defined>                  <Coord-axis>  <Dimname in Datasets>

dimspec    i       1     standard_domain                    x     west_east
dimspec    j       3     standard_domain                    y     south_north
dimspec    k       2     standard_domain                    z     bottom_top

dimspec    lin     2     namelist=num_metgrid_soil_levels   z     num_metgrid_soil_levels
dimspec    snly    2     namelist=num_snow_layers           z     snow_layers
dimspec    l       2     namelist=num_soil_layers           z     soil_layers
dimspec    ulay    2     namelist=num_urban_layers          z     urban_layers
```

In Registry.EM_COMMON

```
state   real   soil_layers    i{lin}j    misc   1  Z  i1  "SOIL_LAYERS"    "SOIL LAYERS"       "cm"
state   real   soil_levels    i{lin}j    misc   1  Z  i1  "SOIL_LEVELS"    "SOIL LEVELS"       "cm"
state   real   st             i{lin}j    misc   1  Z  i1  "ST"             "SOIL TEMPERATURES"
"K"
state   real   sm             i{lin}j    misc   1  Z  i1  "SM"             "SOIL MOISTURES" "m3 m-
3"
```

- Three commonly used types
  1) state
  2) rconfig
  3) package

State Entry

- The entry can be used in an IO stream

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

state  real  tsk   ij    misc   1     -    i01rhud    "TSK"   "SKIN TEMP"
```

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

State Entry

- The entry can be used in an IO stream

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

state  real  tsk   ij    misc   1     -    i01rhud    "TSK"   "SKIN TEMP"
```

- 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 (e.g. wrfinput, wrfout)
- The characters '1' through '9' denote one of the auxiliary input or history I/O streams.
- Double digit streams require "{}" braces: i01{19}{24}

State Entry

- The variable can communicate between nests

```
#       Type Sym  Dims   Use   Tlev  Stag    IO      Dname       Descrip
state   real  tsk   ij    misc   1     -    i01rhud     "TSK"   "SKIN TEMP"
```

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

u – at end of each set of child time steps
d – at instantiation of child domain
f – at beginning of each set of child time steps
s – after each feedback

State Entry

Only variables involved with I/O, communications, packages are required to be state

Variables required from one time step to next

Local variables inside of physics packages are not controlled by the Registry

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

Package Entry

- Elements
  - *Entry*: the keyword "package",
  - *Package name*: the name of the package: e.g. "kesslerscheme"
  - *Associated rconfig choice*: the name of a rconfig variable and the value of that variable that choses **this package**

```
# specification of microphysics options
package   passiveqv     mp_physics==0     -   moist:qv
package   kesslerscheme mp_physics==1     -   moist:qv,qc,qr
package   linscheme     mp_physics==2     -
moist:qv,qc,qr,qi,qs,qg
package   wsm3scheme    mp_physics==3     -   moist:qv,qc,qr
package   wsm5scheme    mp_physics==4     -   moist:qv,qc,qr,qi,qs


# namelist entry that controls microphysics option
rconfig  integer   mp_physics   namelist,physics   max_domains    0
```

Package Entry

- Elements
  - *Package state vars*: unused at present; specify hyphen (-)
  - *Associated* variables: the names of 4D scalar arrays (moist, chem, scalar) and the fields within those arrays this package uses, and the state variables (state:u_gc, ...)

```
# specification of microphysics options
package   passiveqv     mp_physics==0     -   moist:qv
package   kesslerscheme mp_physics==1     -   moist:qv,qc,qr
package   linscheme     mp_physics==2     -
moist:qv,qc,qr,qi,qs,qg
package   wsm3scheme    mp_physics==3     -   moist:qv,qc,qr
package   wsm5scheme    mp_physics==4     -   moist:qv,qc,qr,qi,qs


# namelist entry that controls microphysics option
Rconfig  integer   mp_physics   namelist,physics   max_domains    0
```


Outline

- Registry
- Examples
  1) Add output without recompiling
  2) Add a variable to the namelist
  3) Add an array
  4) Add a passive tracer

Example 1: Add output without recompiling

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

`iofields_filename = "myoutfields.txt"` (MAXDOM)

`io_form_auxhist24 = 2` (choose an available stream)

`auxhist24_interval = 10` (MAXDOM, every 10 minutes)

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

```
+:h:24:RAINC,RAINNC
```

Where "+" means ADD this variable to the output stream, "h" is the history stream, and "24" is the stream number

Example 1: Remove 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: What streams can I use?

- Generally history streams 10 – 24 are OK
- Avoid 22, 23

- Need LOTS more streams?
  - Edit WRFV3/arch/preamble_new

    MAX_HISTORY = 25 <- - - *right now*
  - clean –a, configure, compile, re-run real and wrf

- For production runs, it is more efficient to modify registry directly, and recompile it once.

Outline

- Registry
- Examples
  1) Add output without recompiling
  2) Add a variable to the namelist
  3) Add an array
  4) Add a passive tracer

Example 2: 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
  - Sometimes you need to add your mods to a different Registry for a different dynamical core

Example 2: 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 2: 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 2: 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 2: 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 2: 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
```

Outline

- Registry
- Examples
  1) Add output without recompiling
  2) Add a variable to the namelist
  3) Add an array
  4) Add a passive tracer

Example 3: 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 I1 variable

Example 3: 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 3: 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"
```

34

Example 3: Add an Array

- Always modify Registry.*core_name*_COMMON or Registry.*core_name*, where *core_name* might be EM

```
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"
```

35

Example 3: Add an Array

- Add a new 3D array that is sum of all moisture species, called all_moist, in the Registry.EM_COMMON
  - 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       \
misc          1       -     h             \
"ALL_MOIST"                               \
"sum of all of moisture species"          \
"kg kg-1"
```

36

Example 3: Add an Array

- Registry state variables become part of the derived data structure usually called grid inside of the WRF model.
- WRF model top → integrate → solve_interface → 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 when dereferenced by first_rk_step_part1 for the physics drivers

Example 3: 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 3: Add an Array

- In solve_em, add the new array to the call for the microphysics driver
- Syntax for variable=local_variable is an association convenience
- All 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 3: 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 3: 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
  - Use the correct loop indices

```
!  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 3: 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
```

Outline

- Registry
- Examples
  1) Add output without recompiling
  2) Add a variable to the namelist
  3) Add an array
  4) Add a passive tracer

Tracer Example

Modify Registry for new fields.

Use the "tracer" array with a new 3D component

Use existing NML op

Initialize data in real.

Identify (i,j) location

Spread in "PBL"

Set values in solver.

"Release" per time s

Tracer Example

Registry/Registry.EM add our new field "PLUME" as part of "TRACER" array.

```
#     New tracer for example
state  real  plume  ikjftb  tracer \
     1  -  irhusdf=(bdy_interp:dt) \
     "PLUME"  "Fukushima Tracer"  " "

#     4D arrays need an associated package
package  tracer_test3  tracer_opt==3  - \
     tracer:plume
```

Tracer Example

Modify the real and WRF programs to initialize and continuously re-supply the "PLUME" array

dyn_em/module_initialize_real.F (initial value from real.exe)

dyn_em/solve_em.F (continuous plume in wrf.exe)

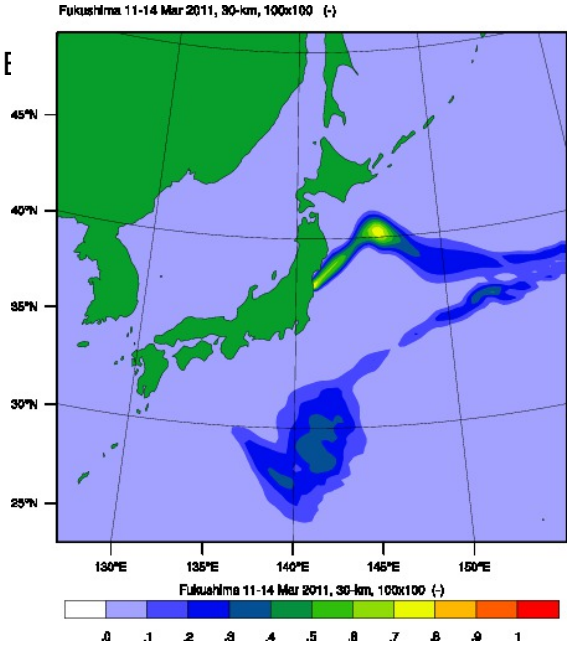
```
!  Add in the Fukushima initial venting.

   IF ( ( its .LE. 50 ) .AND. ( ite .GE. 50 ) .AND. &
        ( jts .LE. 50 ) .AND. ( jte .GE. 50 ) ) THEN
      tracer(50,1:5,50,P_plume) = 1.
   END IF
```

Tracer Example

- Modify the test/em_real/namelist.input file
- Include the new settings for the tracer option required from the Registry file

```
&dynamics
 tracer_opt  = 3, 3, 3,
```

Tracer E



Summary

- WRF Registry
  - What is the registry
  - Keyword syntax
  - Three commonly used types
- Examples
  - Adding and removing output
  - Adding a variable to the namelist
  - Adding an array
  - Adding a tracer