How to Use the WRF Registry

John Michalakes, NCEP

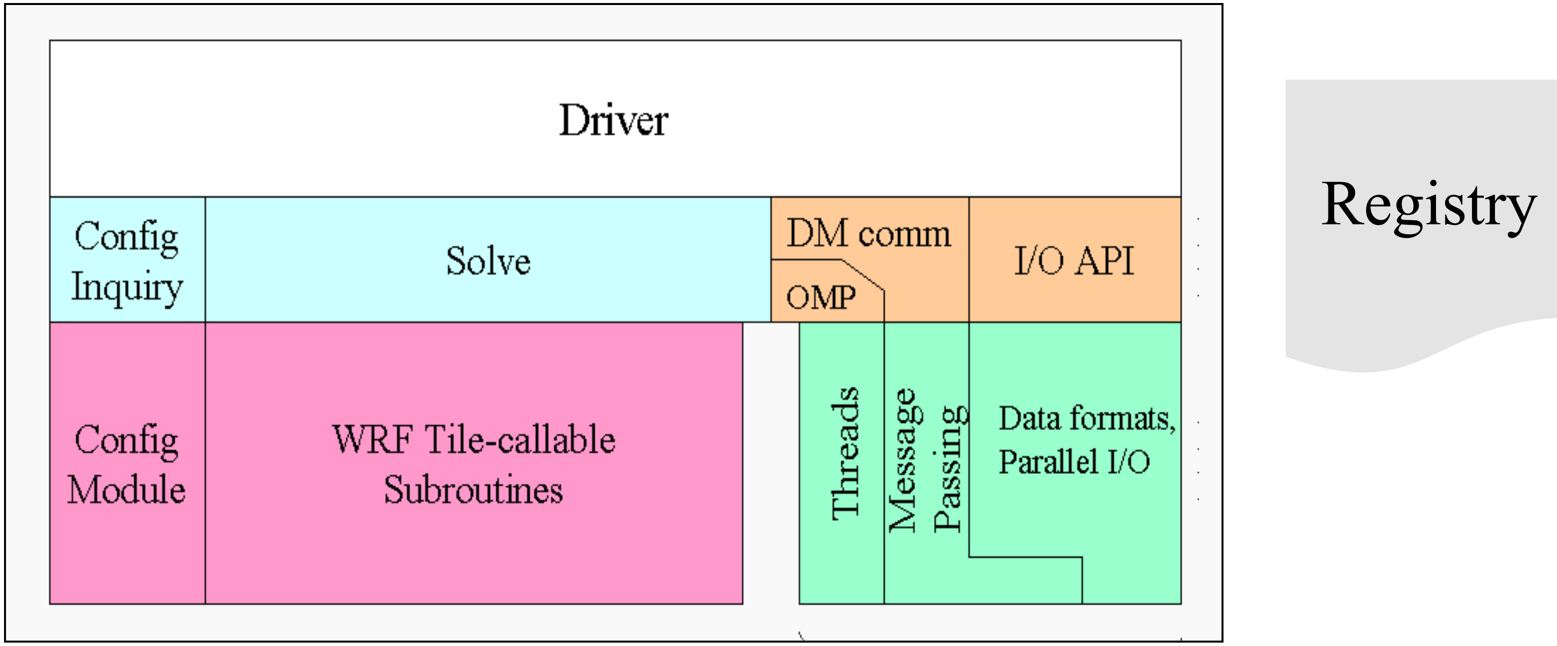
Dave Gill, NCAR

WRF Software Architecture Working Group

Outline

- What is the WRF Registry
- Keyword syntax
- The BIG Three
- Examples
  - Runtime I/O mods
  - Adding a variable to the namelist
  - Adding an array to WRF
  - Compute a diagnostic
  - New physics scheme
  - Passive tracer

WRF Software Architecture



Text based file for real and WRF
Active data dictionary
Used with cpp 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 (else lost changes)

Registry Keywords

- Currently implemented as a text file: Registry/Registry.EM_COMMON
- 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

- Currently implemented as a text file: Registry/Registry.EM_COMMON
- 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 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	-	i01rhusdf	"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     -     i01rhusdf   "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
  - *Units*: Metadata units of the variable
  - *Descrip*: Metadata description of the variable

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	tsk	ij	misc	1	-	i01rhusdf	"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: 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  tsk    ij     misc   1      -     i01rhusdf   "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.
- The characters ‘1’ through ‘9’ denote one of the auxiliary input or history I/O streams.

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  tsk  ij    misc  1     -     i01rhusdf  "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: 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  tsk   ij     misc  1     -      i01rhusdf  "TSK"    "SKIN TEMP"
```

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

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   ncepcloud3    mp_physics==3    -   moist:qv,qc,qr
package   ncepcloud5    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   ncepcloud3    mp_physics==3    -   moist:qv,qc,qr
package   ncepcloud5    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

- Examples
  - 1) Add output without recompiling
  - 2) Add a variable to the namelist
  - 3) Add an array
  - 4) Compute a diagnostic
  - 5) Add a physics package
  - 6) 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: 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)

Outline

- Examples
  - 1) Add output without recompiling
  - 2) Add a variable to the namelist
  - 3) Add an array
  - 4) Compute a diagnostic
  - 5) Add a physics package
  - 6) 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

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

- Examples
  - 1) Add output without recompiling
  - 2) Add a variable to the namelist
  - 3) Add an array
  - 4) Compute a diagnostic
  - 5) Add a physics package
  - 6) 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"
```

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"
```

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       \
dyn_em        1       -     h             \
"ALL_MOIST"                               \
"sum of all of moisture species"          \
"kg kg-1"
```

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

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

- The solve routine calls first_rk_step_part1
- grid is passed in
- No need to pass any variables, such as all_moist

```
!WRF:MEDIATION_LAYER:SOLVER

  CALL first_rk_step_part1( grid , &

  config_flags , &
```

Example 3: Add an Array

- Top of first_rk_step_part1.F
- grid is passed in
- No need to declare any new variables, such as all_moist

```
!WRF:MEDIATION_LAYER:SOLVER

MODULE module_first_rk_step_part1

CONTAINS

  SUBROUTINE first_rk_step_part1 ( grid , &

  config_flags , &
```

Example 3: Add an Array

- In first_rk_step_part1, 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

```
!   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

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

Example 4: Compute a Diagnostic

- Problem: Output global average and global maximum and lat/lon location of maximum for 10 meter wind speed in WRF
- Steps:
  - Modify solve to compute wind-speed and then compute the local sum and maxima at the end of each time step
  - Use reduction operations built-in to WRF software to compute the global qualities
  - Output these on one process (process zero, the “monitor” process)

Example 4: Compute a Diagnostic

- Compute local sum and local max and the local indices of the local maximum

```
   --- File: dyn_em/solve_em.F  (near the end) ---

! Compute local maximum and sum of 10m wind-speed
   sum_ws = 0.
   max_ws = 0.
   DO j = jps, jpe
     DO i = ips, ipe
       wind_vel = sqrt( grid%u10(i,j)**2+ grid%v10(i,j)**2 )
       IF ( wind_vel .GT. max_ws ) THEN
          max_ws = wind_vel
          idex = i
          jdex = j
       ENDIF
       sum_ws = sum_ws + wind_vel
     ENDDO
   ENDDO
```

Example 4: Compute a Diagnostic

- Compute global sum, global max, and indices of the global max (WRF intrinsics)

```
! Compute global sum
   sum_ws = wrf_dm_sum_real ( sum_ws )

! Compute global maximum and associated i,j point
   CALL wrf_dm_maxval_real ( max_ws, idex, jdex )
```

Example 4: Compute a Diagnostic

- On the process that contains the maximum value, obtain the latitude and longitude of that point; on other processes set to an artificially low value.
- The use parallel reduction to store that result on every process

```
   IF ( ips .LE. idex .AND. idex .LE. ipe .AND.  &
        jps .LE. jdex .AND. jdex .LE. jpe ) THEN
      glat = grid%xlat(idex,jdex)
      glon = grid%xlong(idex,jdex)
   ELSE
      glat = -99999.
      glon = -99999.
   ENDIF

! Compute global maximum to find glat and glon
   glat = wrf_dm_max_real ( glat )
   glon = wrf_dm_max_real ( glon )
```

Example 4: Compute a Diagnostic

- Output the value on process zero, the “monitor”

```
! Print out the result on the monitor process
   IF ( wrf_dm_on_monitor() ) THEN
      WRITE(outstring,*)'Avg. ',sum_ws/((ide-ids+1)*(jde-jds+1))
      CALL wrf_message ( TRIM(outstring) )
      WRITE(outstring,*)'Max. ',max_ws,' Lat. ',glat,&
                                       ' Lon. ',glon
      CALL wrf_message ( TRIM(outstring) )
   ENDIF
```

Example 4: Compute a Diagnostic

- Output from process zero of a multi-process run

```
     --- Output file: rsl.out.0000 ---
     . . .
   Avg.     5.159380
   Max.     15.09370      Lat.     37.25022      Lon.    -67.44571
Timing for main: time 2000-01-24_12:03:00 on domain   1:     8.96500 elapsed secs.
   Avg.     5.166167
   Max.     14.97418      Lat.     37.25022      Lon.    -67.44571
Timing for main: time 2000-01-24_12:06:00 on domain   1:     4.89460 elapsed secs.
   Avg.     5.205693
   Max.     14.92687      Lat.     37.25022      Lon.    -67.44571
Timing for main: time 2000-01-24_12:09:00 on domain   1:     4.83500 elapsed secs.
    . . .
```

Outline

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

Example 5: Input periodic SSTs

- Add a new physics package with time varying input source to the model
- This is how we could supply a time varying value to the model for a field that is traditionally fixed
- Example is sea surface temperature

Example 5: Input periodic SSTs

- Problem: adapt WRF to input a time-varying lower boundary condition, e.g. SSTs, from an input file for a new surface scheme
- Given: Input file in WRF I/O format containing 12-hourly SST's
- Modify WRF model to read these into a new state array and make available to WRF surface physics

Example 5: Input periodic SSTs

- Steps
  - Add a new state variable and definition of a new surface layer package (that will use the variable) to the Registry
  - Add to variable stream for an unused Auxiliary Input stream
  - Adapt physics interface to pass new state variable to physics
  - Setup namelist to input the file at desired interval

Example 5: Input periodic SSTs

- Add a new state variable to Registry/Registry.EM_COMMON and put it in the variable set for input on Auxiliary Input Stream #4

```
#     type  symbol dims use  tl stag  io    dname      description        units
state real  nsst   ij   misc  1  -    i4h  "NEW_SST" "Time Varying SST" "K"
```

- Also added to History and Restart
- Result:
  - 2-D variable named grid%**nsst** defined and available in solve_em
  - Dimensions: ims:ime, jms:jme
  - Input and output on the AuxInput #4 stream will include the variable under the name NEW_SST

Example 5: Input periodic SSTs

- Pass new state variable to surface physics

```
      --- File: dyn_em/module_first_rk_step_part1.F ---

  CALL surface_driver(                                                &
              . . .
! Optional
 &        ,QV_CURR=moist(ims,kms,jms,P_QV), F_QV=F_QV                 &
 &        ,QC_CURR=moist(ims,kms,jms,P_QC), F_QC=F_QC                 &
 &        ,QR_CURR=moist(ims,kms,jms,P_QR), F_QR=F_QR                 &
 &        ,QI_CURR=moist(ims,kms,jms,P_QI), F_QI=F_QI                 &
 &        ,QS_CURR=moist(ims,kms,jms,P_QS), F_QS=F_QS                 &
 &        ,QG_CURR=moist(ims,kms,jms,P_QG), F_QG=F_QG                 &
 &        ,NSST=grid%nsst                                             & ! new
 &        ,CAPG=grid%capg, EMISS=grid%emiss, HOL=hol,MOL=grid%mol     &
 &        ,RAINBL=grid%rainbl,SR=grid%em_sr                           &
 &        ,RAINNCV=grid%rainncv,REGIME=regime,T2=grid%t2,THC=grid%thc &
               . . .
```

Example 5: Input periodic SSTs

- Add new variable nsst to Physics Driver in Mediation Layer

```
     --- File: phys/module_surface_driver.F ---

SUBROUTINE surface_driver(                                          &
         . . .
            !  Other optionals (more or less em specific)
  &             ,nsst                                                &
  &             ,capg,emiss,hol,mol                                  &
  &             ,raincv,rainbl,regime,t2,thc                         &
  &             ,qsg,qvg,qcg,soilt1,tsnav                            &
  &             ,smfr3d,keepfr3dflag                                 &
        . . .
                                                                     ))
        . . .
REAL, DIMENSION( ims:ime, jms:jme ), OPTIONAL, INTENT(INOUT)::   nsst
```

- By making this an "Optional" argument, we preserve the driver's compatibility with other cores and with versions of WRF where this variable hasn't been added.

Example 5: Input periodic SSTs

- Add call to Model-Layer subroutine for new physics package to Surface Driver

```
        --- File: phys/module_surface_driver ---

!$OMP PARALLEL DO   &
!$OMP PRIVATE ( ij, i, j, k )
   DO ij = 1 , num_tiles
     sfclay_select: SELECT CASE(sf_sfclay_physics)

       CASE (SFCLAYSCHEME)
          . . .
       CASE (NEWSFCSCHEME)  ! <- This is defined by the Registry "package" entry

         IF (PRESENT(nsst))  THEN
            CALL NEWSFCCHEME(                                        &
                nsst,                                                &
                ids,ide, jds,jde, kds,kde,                           &
                ims,ime, jms,jme, kms,kme,                           &
                i_start(ij),i_end(ij), j_start(ij),j_end(ij), kts,kte    )
         ELSE
           CALL wrf_error_fatal('Missing argument for NEWSCHEME in surface driver')
         ENDIF
          . . .
     END SELECT sfclay_select
   ENDDO
!$OMP END PARALLEL DO
```

- Note the PRESENT test to make sure new optional variable nsst is available

Example 5: Input periodic SSTs

- Add definition for new physics package NEWSCHEME as setting 4 for namelist variable sf_sfclay_physics

```
rconfig   integer   sf_sfclay_physics   namelist,physics   max_domains    0

package   sfclayscheme   sf_sfclay_physics==1        -              -
package   myjsfcscheme   sf_sfclay_physics==2        -              -
package   gfssfcscheme   sf_sfclay_physics==3        -              -
package   newsfcscheme   sf_sfclay_physics==4        -              -
```

- This creates a defined constant NEWSFCSCHEME and represents selection of the new scheme when the namelist variable sf_sfclay_physics is set to '4' in the namelist.input file
- `clean -a` and recompile so code and Registry changes take effect

Example 5: Input periodic SSTs

- Setup namelist to input SSTs from the file at desired interval

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

&time_control
    . . .
 auxinput4_inname       = "sst_input"
 auxinput4_interval_h  = 12
  . . .
/

    . . .
&physics
 sf_sfclay_physics   = 4, 4, 4
    . . .
/
```

- Run code with sst_input file in run-directory

Example 5: Input periodic SSTs

- A few notes…
  - The read times and the time-stamps in the input file must match exactly
  - We haven't done anything about what happens if the file runs out of time periods (the last time period read will be used over and over again, though you'll see some error messages in the output if you set debug_level to be 1 or greater in namelist.input)
  - We haven't said anything about what generates sst_input

Outline

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

Tracer Example

Modify Registry for new fields.

- Use the "tracer" array with a new 3D component
- Use existing NML option

Initialize data in real.

- Identify (i,j) location
- Spread in "PBL"

Set values in solver.

- "Release" per time step

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)

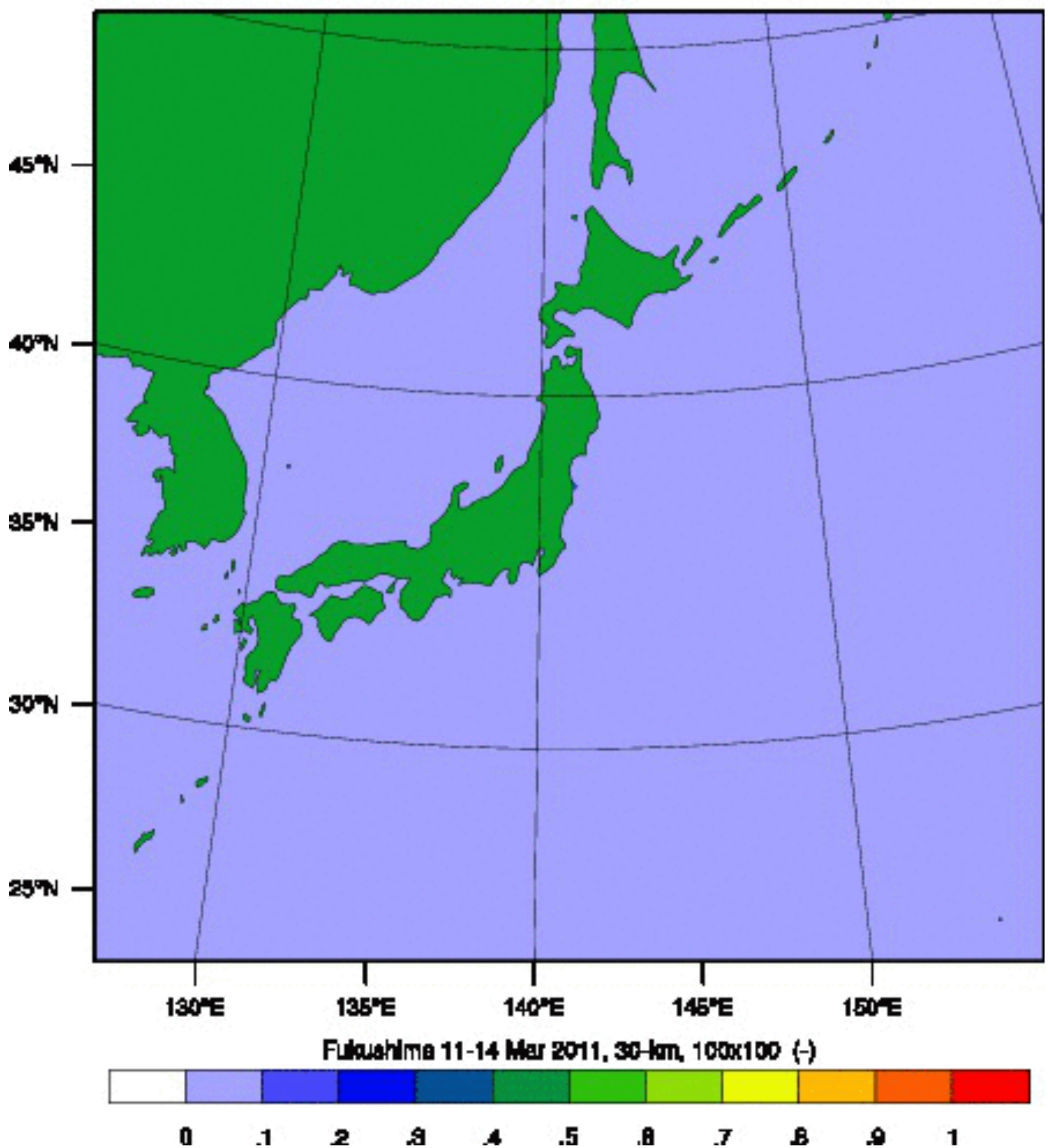
```
!  Add in the Fukushima initial venting.

   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
```

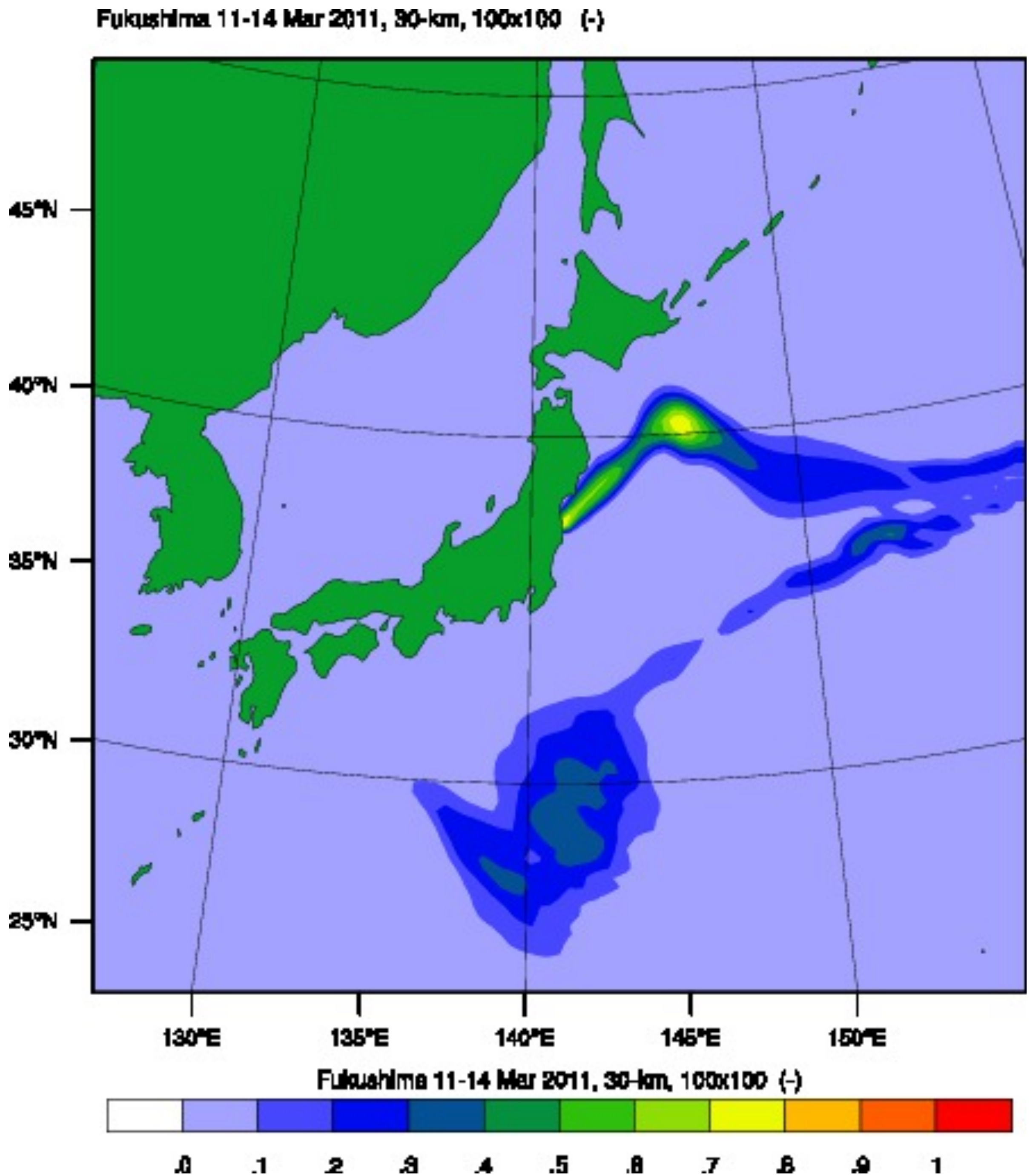
Tracer Example

- 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,
```

Fukushima 11-14 Mar 2011, 30-km, 100x100 (-)




Fukushima 11-14 Mar 2011, 30-km, 100x100 (-)
45°N
40°N
35°N
30°N
25°N
130°E
135°E
140°E
145°E
150°E
Fukushima 11-14 Mar 2011, 30-km, 100x100 (-)
.0
.1
.2
.3
.4
.5
.6
.7
.8
.9
1