Better Practices and Hints
WRF Source and Infrastructure

Dave Gill

Better Practices and Hints for WRF

With dozens of contributors, the presumed *Better* Practices and Hints have evolved to what we currently prefer …

Focus for this talk:

- Adding or modifying compilable WRF source code
- Adding to or modifying the WRF Registry
- Debugging

Adding or modifying compilable WRF source code

Adding or modifying compilable WRF source code

Does WRF care about free *vs* fixed form?

The WRF source code is mostly a free-formatted Fortran code base.

The configure.wrf file has exlicit rules for the free and fixed codes.

The WRF build system assumes that all of user contributed code will compile as free-formatted.

Adding or modifying compilable WRF source code

Is there a Fortran standard to which the user should adhere?

There are a few Fortran 2003 capabilities included, and most recent compilers support these.

Do not attempt to utilize coarrays or other more exotic additions to the Fortran standard.

Compilers vary in there support for newer capabilities, both being able to compile them, and to use them efficiently.

Adding or modifying compilable WRF source code

How should the call to my new physics package be done?

Look in the specific driver for examples of existing calls. For example phys/module_radiation_driver.F

Adding or modifying compilable WRF source code

How should the call to my new physics package be done?

All top level physics routines are called with a 3d block of data.

18 dimensions are always passed:

- I, J, K dimensions
- domain, memory, and computational sizes
- starting and ending

Adding or modifying compilable WRF source code

How should the call to my new physics package be done?

```
CALL cal_cldfra1(CLDFRA,qv,qc,qi,qs,          &
                 F_QV,F_QC,F_QI,F_QS,t,p,     &
                 F_ICE_PHY,F_RAIN_PHY,        &
                 ids,ide, jds,jde, kds,kde,   &
                 ims,ime, jms,jme, kms,kme,   &
                 its,ite, jts,jte, kts,kte    )
```

Adding or modifying compilable WRF source code

Where does WRF assume that the values inside of the physics schemes are located?

The physics schemes are column oriented, no communications are required top to bottom.

The values are located at mass points.

Some variables are located on full eta level (usually denoted with the cryptic convention “8w”), but most 3d variables are located on the computational half layer locations.

Adding or modifying compilable WRF source code

If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

All variables in the Registry (state + namelist options) are in the derived data structure “grid”.

When “grid” is available, the new variable does not need to be dereferenced from the structure.

The calls to the drivers in module_first_rk_step_part1.F exhibit the required dereferencing.

Adding or modifying compilable WRF source code

If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

```
CALL radiation_driver(     &
   ACFRCV=grid%acfrcv    , &
   ACFRST=grid%acfrst    , &
   ALBEDO=grid%albedo    , &
```

Adding or modifying compilable WRF source code

If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

The call to the specific driver needs to have the new variable explicitly passed in from module_first_rk_step_part1.F (or from the solver for the microphysics routines).

User modifications are then required in all deeper routines.

Adding or modifying compilable WRF source code

If there is a new variable that needs to be communicated, how is that set up in WRF?

All communications in WRF are a combination of two items: manual inclusion of compilable source code, and manual inclusion of communications information in the Registry.

The source modifications “cpp include” a file into the source prior to compilation.

Adding or modifying compilable WRF source code

If there is a new variable that needs to be communicated, how is that set up in WRF?

The developer may choose to communicate the variables immediately after the computation is performed to manufacture the new variable, or wait until the new variable's halo is needed.

Adding or modifying compilable WRF source code

If there is a new variable that needs to be communicated, how is that set up in WRF?

```
#ifdef DM_PARALLEL
#  include "PERIOD_BDY_EM_A.inc"
#endif
```

Adding or modifying compilable WRF source code

How do you access a particular hydrometeor from the 4d array moist?

The name associated with the variable defined in the Registry is used to construct a Fortran PARAMETER value.

This integer index should always be used to refer to the particular 3d array.

Adding or modifying compilable WRF source code

How do you access a particular hydrometeor from the 4d array moist?

Registry, first few parts of the QVAPOR line:

```
state    real     qv     ikjftb    moist
```

Source code:

```
qvf = 1. + rvovrd*moist(i,k,j,P_QV)
```

Adding or modifying compilable WRF source code

How do you access a particular hydrometeor from the 4d array moist?

Loops over the 4d arrays should always begin and end with the WRF specific starting values:

```
DO im = PARAM_FIRST_SCALAR, num_3d_m
   qtot = qtot + moist(i,k,j,im)
ENDDO
```

Adding or modifying compilable WRF source code

How do you access a particular hydrometeor from the 4d array moist?

These automatically generated values are inside module_state_description.

When these generated indexes are required for new code, USE module_state_description.

Adding or modifying compilable WRF source code

With modern Fortran, how do I get information from the module?

A "use association" is employed in WRF.

To restrict the number of symbol names that are shared, the WRF code tends to restrict the variables requested with the ONLY clause.

Mostly this "ONLY clause" is added to keep compilers from complaining about source code being "too complex" when the used module is large.

Adding or modifying compilable WRF source code

With modern Fortran, how do I get information from the module?

```
USE module_configure, ONLY : grid_config_rec_type
USE module_driver_constants
USE module_machine
USE module_tiles, ONLY : set_tiles
```

Adding or modifying compilable WRF source code

With modern Fortran, how do I get information from the module?

Typically when adding in communications or new physics packages, the USE statements need to be amended to include the new Registry information.

Adding or modifying compilable WRF source code

If there are known restrictions for packages, how can that information be used at model initialization?

There are two mechanisms in WRF for handling error checking for the physics schemes:

phys/module_physics_init.F

share/module_check_a_mundo.F

Adding or modifying compilable WRF source code

If there are known restrictions for packages, how can that information be used at model initialization?

The tests in physics_init are more aligned with modifying 2d and 3d arrays depending on the values of namelist settings or other 2d and 3d arrays. Initializations for each domain take place.

To avoid OpenMP race conditions, this is a much better way to fix zeroed-out variables.

Adding or modifying compilable WRF source code

If there are known restrictions for packages, how can that information be used at model initialization?

The purpose for check_a_mundo is to stop imcompatible namelist options. If a user knows that a certain scheme is only set up to work with one type of PBL, then that needs to be included.

Adding or modifying compilable WRF source code

What does the WRF model mean by “restart”, and how does it impact a developer?

A restart in WRF allows a model simulation to continue from an interrupted state, AND to produce bit-wise identical results to those generated from a non-interrupted simulation.

Developers need to provide information as to which variables need to be saved for a restart run.

Adding or modifying compilable WRF source code

What does the WRF model mean by “restart”, and how does it impact a developer?

The restart variables are explicitly listed in the Registry.

```
state      real    rimi     ikj     misc \
1      -       irh   \
"RIMI"     "riming intensity" \
"fraction"
```

Adding or modifying compilable WRF source code

What does the WRF model mean by “restart”, and how does it impact a developer?

The physics_init routine needs to avoid resetting restarted variables, which requires user modification.

```
IF(.not.restart)THEN
  !-- initialize common variables
  IF(PRESENT(rliq)) THEN
    rliq(:,:) = 0.0
  ENDIF
ENDIF
```

Adding or modifying compilable WRF source code

Will the WRF community sing my praises if I, as a developer, include lots of inline documentation?

YES!

Adding or modifying compilable WRF source code

Is there a mechanism to output the namelist options that are being developed?

The file share/output_wrf.F handles the metadata output:

```
ibuf(1) = config_flags%e_we - config_flags%s_we + 1
CALL wrf_put_dom_ti_integer ( fid , &
   'WEST-EAST_GRID_DIMENSION' ,  ibuf , 1 , ierr )
```

Adding or modifying compilable WRF source code

Is there a mechanism to output the namelist options that are being developed?

Routines exist to output integer, real, logical, and character strings.

Adding or modifying compilable WRF source code

What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

There are a few routines in WRF that handle these types of capabilities for most traditional data types (real, double, integer).

Adding or modifying compilable WRF source code

What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

There are a few routines in WRF that handle these types of capabilities for most traditional data types (real, double, integer).

Adding or modifying compilable WRF source code

What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

```
lat1 = wrf_dm_min_real ( lat1 )
```

Please see example #3 for a more complete list and examples of usage:

http://www2.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Registry_2.pdf

Adding or modifying compilable WRF source code

How about initializations that can only be handled on the master node?

Even for serially-built code, the following is defined:

```
IF ( wrf_dm_on_monitor() ) THEN
```

Adding or modifying compilable WRF source code

How does info get from the master node to the other processors?

Again for native data types, a variable (and the number of words) can be broadcast to all of the processors from the master.

```
CALL wrf_dm_bcast_integer(nt,1)
```

Adding or modifying compilable WRF source code

If information is in the namelist, how is it accessed inside the code?

There are three methods to get namelist information:

- grid
- config_flags
- subroutines

Adding or modifying compilable WRF source code

If information is in the namelist, how is it accessed inside the code?

Any time the “grid” structure is present, the namelist option may be dereferenced as an existing field in the derived structure:

```
p_top_requested = grid%p_top_requested
```

Adding or modifying compilable WRF source code

If information is in the namelist, how is it accessed inside the code?

Similarly, the derived structure config_flags holds the namelist information for the current grid being processed:

```
IF ( config_flags%spec_bdy_width .GT. &
     flag_excluded_middle ) THEN
```

Adding or modifying compilable WRF source code

If information is in the namelist, how is it accessed inside the code?

The WRF code automatically builds two subroutines for each namelist variable, a “get” and a “set” subroutine. Mostly, developers are interested in the “get” option. Argument #1 is which domain, and argument #2 is the local returned value.

```
CALL nl_get_base_pres  ( 1 , p00 )
```

Adding or modifying compilable WRF source code

What are the available options for outputting debug print information?

Because not all print buffers are guaranteed to be flushed on an error exit, it is better to use WRF supplied print-out functions.

Also, use the WRF provided fatal error function instead of a Fortran STOP statement.

Adding or modifying compilable WRF source code

What are the available options for outputting debug print information?

```
CALL wrf_debug ( 200 , ' call end of solve_em' )
CALL wrf_message('ndown: using namelist constants')
CALL wrf_error_fatal( ’Use km_opt=2 with sfs_opt=2’)
```

Adding or modifying compilable WRF source code

If a developer wants an event to occur every so often, how is that accomplished?

Be wary of a simple

MOD (current_time , some_interval) == 0

set up. For large values of current time, the statement may eventually never be true again. For a fixed time step, the integer number of time steps might be preferable:

```
IF (mod(itimestep,STEPFG) .eq. 0) THEN
```

Adding or modifying compilable WRF source code

What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

First, this is required due to the two different dynamical cores inside of WRF, and even for ARW the DA and Chem codes do not need all variables. To allow the physics schemes to work with both cores (and the Chem and DA options), some variables are considered optional because they are not present at all times.

Adding or modifying compilable WRF source code

What is supposed to happen with OPTIONAL variables and the CPP ifdef’ing?

For a new scheme, if only a “few” variables are to added to both cores, it is reasonable to add the variables to both the ARW and NMM Registry files (similarly, the DA and Chem Registry files).

If LOTS of variables are to be added, it is better to go the OPTIONAL variable route.

Adding or modifying compilable WRF source code

What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

The variables that are only required for one of the build options are ifdef'ed out in the calling routine (for example the first_rk_step_part1 file

```
#ifdef WRF_CHEM
     & ,CHEM=chem,chem_opt=config_flags%chem_opt
&
#endif
```

Adding or modifying compilable WRF source code

What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

Always add new variables to a Registry package to minimize the model's memory footprint. The variables are allocated, but only with a (1,1,1) size.

When using OPTIONAL arguments, always test if the variable is PRESENT before using.

Adding or modifying compilable WRF source code

What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

```
DO j=j_start(ij),j_end(ij)
DO i=i_start(ij),i_end(ij)
  IF ( PRESENT( rainshv )) THEN
    RAINBL(i,j) = RAINBL(i,j) + RAINSHV(i,j)
  END IF
ENDDO
ENDDO
```

Adding or modifying compilable WRF source code

What are the usual ifdef syntaxes that are to be used?

To avoid the situation where a compile-time option is set to zero (where the intent was to turn the option OFF), the WRF ifdef's test on the number "1".

Adding or modifying compilable WRF source code

What are the usual ifdef syntaxes that are to be used?

```
#if ( DA_CORE == 1 )

#if ( WRF_CHEM == 1 )

#if ( NMM_CORE == 1 )

#if ( EM_CORE == 1 )
```

Adding or modifying compilable WRF source code

What type of communications are allowed between columns in the physics schemes?

By default, the physics schemes are column oriented, with no impact permitted from neighbors.

This means NO horizontal differences or horizontal averaging inside the physics schemes.

Adding to or modifying the WRF Registry

Adding to or modifying the WRF Registry

What is the WRF Registry?

The WRF Registry is an active data dictionary.

It is a text-based file that is user-modifiable.

Every variable used with I/O, communications, namelist option is in the Registry.

All associations of variables with physics schemes is handled by the Registry.

Adding to or modifying the WRF Registry

What is the WRF Registry?

The text-based file is read by a program.

This registry program manufactures include files that are CPP #include'd during the WRF build process.

More than 300 thousand lines of automatically generated code are included in the WRF source code via the registry program.

Adding to or modifying the WRF Registry

What are the different types of model variables in the Registry?

Most users are concerned with the gridded data or with the namelist variables. The Registry handles both of these.

The gridded data is either “state” (available throughout the duration of the simulation) or the data is “i1” (tendency variables that pop off the stack at the conclusion of each time step).

Adding to or modifying the WRF Registry

What are the different types of model variables in the Registry?

Please see for more information on the Registry:

`http://www2.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Registry_1.pdf`

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

There are multiple streams (think of these as separate unit numbers) for input and output.

Each variable may be in zero or more streams.

The WRF naming convention for the streams:

i => input

h => history

r => restart

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

For input and history, the default stream is “0”.

A stream specification of “ih” assumes the field is in the input stream and will be output to the WRF history file.

Numerals are added after the characters “i” or “h” to indicate additional (nonstandard) streams for the fields.

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

Once explicit stream numbers are specified, the “zero” stream must also be specifically requested.

Streams with more than one digit, for example stream #14, would be surrounded by “{}”

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

First few entries for the eta levels:

```
state real znu   k  dyn_em 1 - irh
state real znw   k  dyn_em 1 Z i0rh
```

Note that irh and i0rh are identical specifications.

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

First few entries for the eta levels:

```
state real znu   k  dyn_em 1 - irh
state real znw   k  dyn_em 1 Z i0rh
```

Note that irh and i0rh are identical specifications.

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

The 2-m temperature has input from real (i0), input from metgrid (i1), output to the default history file (h0), and output to an auxiliary stream (h{23}):

```
state real T2    ij misc   1 - i01rh0{23}
```

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

All variables involved with I/O are required to be state.

The state variables may be real, double, integer, character, or logical.

Variables for I/O must be 0d, 1d, 2d, 3d, or part of a known 4d amalgamation.

Only one time slice of two-time-level fields is output.

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

The Registry is not involved in the actual format of the input or output data.

The format, frequency, name of the file, etc are all run-time options (though the namelist options controlling those capabilities are defined in the Registry).

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

Only use an “i” for variables that are input. For example, convective precipitation is not input from the real program, and should not have an “i”.

Similarly, developers tend to think every variable that was used is vital. Judiciously select those that will be given an “h” designation.

Adding to or modifying the WRF Registry

How is I/O handled in the Registry?

The “r” designator is mandatory for fields that are required to manufacture an identical simulation, when comparing a restart run to a non-restart run.

Including an “r” for non-mandatory fields makes the restart file large and the associated I/O slow, but otherwise has no forecast impact.

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

The same block of information controlling the I/O has a few keywords that control the nesting.

- u => feedback up to parent mesh
- d => horizontally interpolate down to child domain
- f => lateral boundary forcing
- s => smoothing on CG in area of FG

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

The “u”, “d”, and “f” options are able to use a default for most continuous variables (though the horizontal staggering is important).

Developers may associate a new subroutine with a new physics variable, though this is not too common.

Almost all of the lateral boundary forcing is the dynamics variables, with no usage for the physics variables.

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

As with most Registry items, it is usually safest for a developer to copy a similar (and existing) Registry line for the initial idea for a new variable.

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

Developers handling land surface fields must be concerned with masking.

An average across the spatial extent of a parent cell might include both water and land points from the child, which would feedback garbage to the parent.

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

While the mnemonics of “u” and “d” refer to “up” and “down”, respectively, the WRF nesting code is general.

d => once only, at the start of the model simulation

u => child to parent, at the end of each child set of time steps

f => parent to child, at the start of each set of child time steps

Adding to or modifying the WRF Registry

How is nesting handled in the Registry?

For example, CG SST is handed to the FG at each parent time step via an “f” option (subroutine: c2f_interp):

```
state    real     OM_TMP \
i{nocnl}j         misc      1     Z \
i012rhdu=(copy_fcnm) \
f=(c2f_interp:grid_id)      \
"OM_TMP" "temperature"    "k"
```

Adding to or modifying the WRF Registry

How is communication handled in the Registry?

There are three kinds of communications possible with WRF:

- halo => next door neighbor
- period => west-east or south-north exchange
- transpose => largely for FFTs

Most developers are only concerned with halo communications.

Adding to or modifying the WRF Registry

How is communication handled in the Registry?

The halo comms are specified for a list variables, and the size of the stencil for each those variables.

Please see for more information on WRF stencils:

`http://www2.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Software.pdf`

Adding to or modifying the WRF Registry

How is communication handled in the Registry?

Overspecifying the size of the stencil has no ill effects on results, it is just a performance sink.

The same communication pattern may “used” inside of WRF multiple times.

Adding to or modifying the WRF Registry

How does the Registry help with memory management?

The Registry offers a "package" option which associates state variables with particular namelist options.

Developers should include this for their schemes.

Variables are allocated only with 1 word of space (for example: (1,1,1) for a 3d array, and (1,1) for a 2d array).

Adding to or modifying the WRF Registry

How does the Registry help with memory management?

The package option is able to handle conditional namelist settings through the use of derived namelist settings.

The data used from metgrid by the real program is not required by the WRF model, so it is in a package controlled by a derived namelist variable.

Adding to or modifying the WRF Registry

How does the Registry help with memory management?

```
rconfig   integer use_wps_input    \
derived     1       0
```

```
package   realonly    use_wps_input==1   -   \
state:u_gc,v_gc,…
```

Debugging

Debugging

What are simple recommendations for trying to debug a problem in WRF?

The WRF build system allows the user to configure the model to run with many compiler-supplied error trapping systems activated.

```
   ./configure –D
```

This executable will run VERY slowly.

Debugging

What are simple recommendations for trying to debug a problem in WRF?

Try to track down problems in big domains by simplifying:

- smaller domains
- single processor
- remove physics options sequentially
- short forecasts through use of restart

Debugging

When does NCAR wants to be contacted?

When a standard model set up fails, we want to know.

“Standard” is a recent release running with reasonable settings, and typical input data that we frequently run.

Debugging

When does NCAR NOT want to be contacted?

A developer wrote code, and now the WRF model fails when the option is turned on.

A developer wrote code, and now the WRF model fails even when the option is turned off.

=> NCAR WRF user support cannot conduct research for a developer.

Debugging

What are typical failure modes for WRF?

Bad initial conditions

The model simulation fails quickly (first few time steps).

If the time step is OK, look for DRAMATICALLY bad fields, such as from a flag value, not an actual physically meaningful value.

Debugging

What are typical failure modes for WRF?

Bad initial conditions

Too many, too few vertical levels.

Poorly distributed vertical levels (let the real program figure them out).

Debugging

What are typical failure modes for WRF?

Bad initial conditions

If any (i,j) info is provided by the WRF model, use a visual tool (ncview) to look at that location for these fields: MU, MUB, U, V, T, PH, PHB, QVAPOR, W.

For 3d arrays, look top to bottom.

The masked fields may be problematic at the initial time: TSLB, SMOIS, SEAICE, SST.

Debugging

What are typical failure modes for WRF?

Model is unstable early on

The CFL violations are reported for values > 2.

Values that are larger will kill the WRF simulation.

Early CFL problems MIGHT be alleviated with a shorter time step.

Modify solve_em.F to force the RK and the sound loop to have only one iteration to localize the problem.

Debugging

What are typical failure modes for WRF?

Model is unstable early on

Again, early on, most troubles stem from the IC.

Regardless of the location of the failure message (cumulus, radiation, land surface), review closely the IC file.

The problem, for an early failure, is unlikely to be due to a problem in the radiation scheme, for example.

Debugging

What are typical failure modes for WRF?

Model is unstable later in the simulation

Usually, shortening the time step is not that helpful.

Take care to notice if the reported CFL violations in the rsl files are fatal, or just “business as usual” and the model has recovered with vertical velocity damping.

Later-in-the-simulation failures are hard to solve.

Debugging

What are typical failure modes for WRF?

Model is unstable later in the simulation

Is the failure reproducible – on a re-run does the WRF model fail exactly the same in exactly the same place.

Reproducible failures allow a restart file to get a short simulation to test.

If the restarted simulation also successfully fails, then a recompiled code with error trapping activated may help out.

Debugging

What are tools that NCAR WRF user support uses for debugging the model when it fails?

With netcdf output, a number of simple visual tools are available: ncview, ncl.

If the model shows significant sensitivity to physical parameterization settings, ncdiff for a few variables might be helpful.

Variables to particularly consider: MU, MUB, U, V, W, T, PH, PHB, QVAPOR, TSLB, SMOIS

Debugging

What are tools that NCAR WRF user support uses for debugging the model when it fails?

Running with different compilers (or even different versions) is sometimes helpful.

When a failure occurs:

- Does the model work with different ICs
- Same IC source, different day
- Different physics

Debugging

What are tools that NCAR WRF user support uses for debugging the model when it fails?

Looking for when the code does not fail is helpful:

- Domain size
- Number of procs
- Different compiler
- Different case with same namelist
- Same case with different namelist settings