The Common Community Physics Package (CCPP)
- developer session -

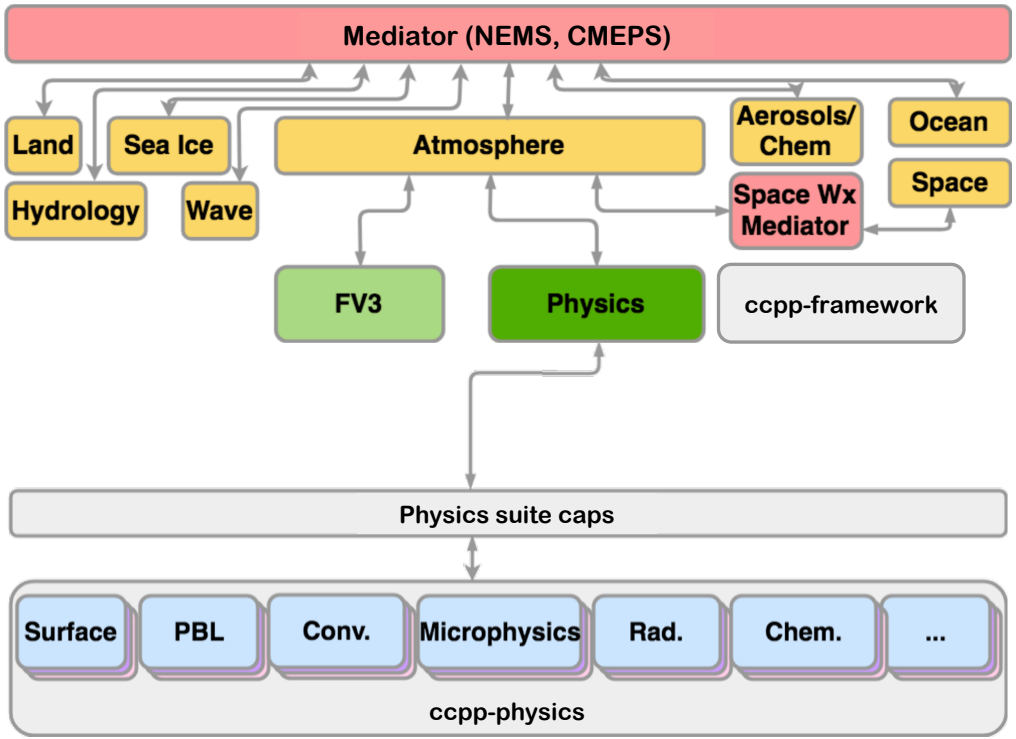
Dom Heinzeller[1,3,4], Grant Firl[2,3], Ligia Bernardet[1,3],
Laurie Carson[2,3], Man Zhang[1,3,4], Julie Schramm[2,3] , Linlin Pan[1,3,4]

[1] NOAA/ESRL Global Systems Laboratory
[2] NCAR Research Applications Laboratory
[3] Developmental Testbed Center
[4] University of Colorado Cooperative Institute for Research in Environmental Sciences

Warmup/recap: CCPP's role and function in a modeling system, philosophy and standardization

CCPP is the infrastructure for physics development

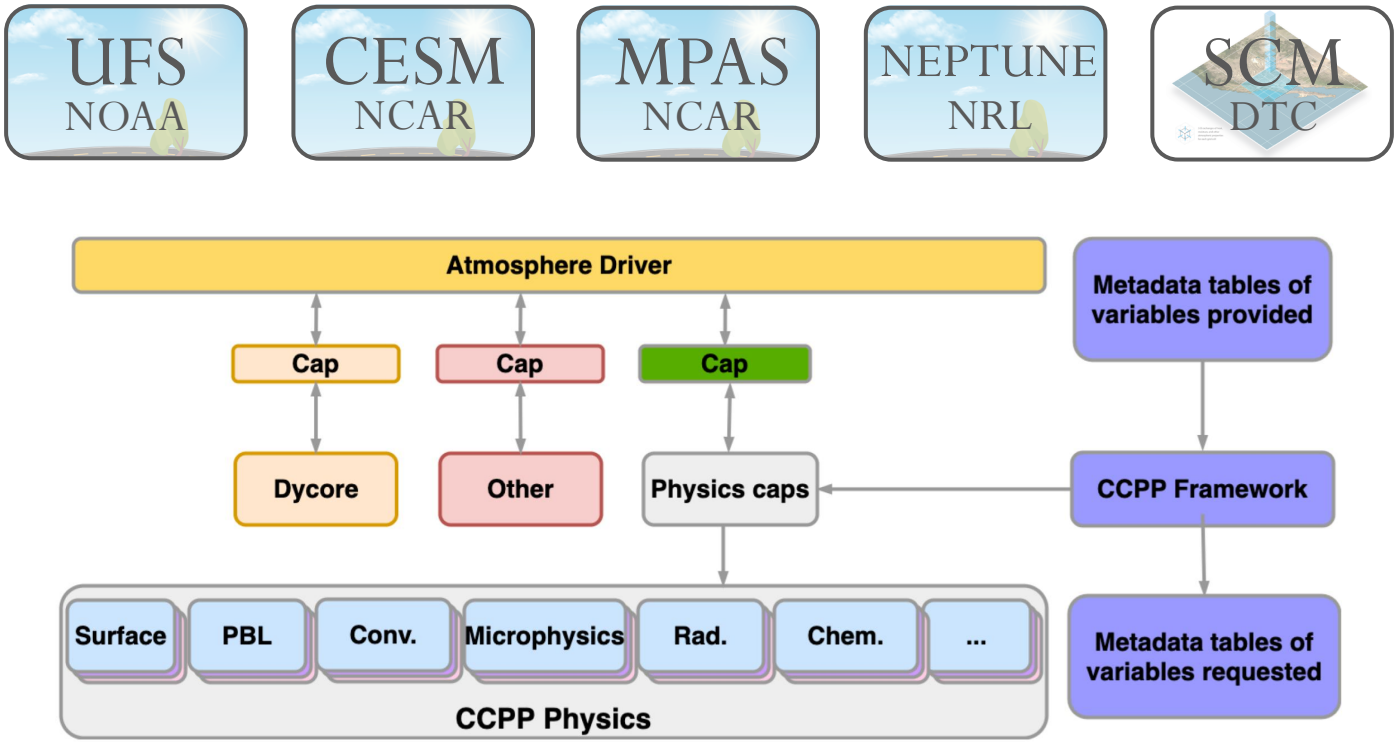
… facilitate the improvement of physical parameterizations and their transition from research to operations by enabling the community to participate in the development and testing …



Common Community Physics Package (CCPP)

https://github.com/NCAR/ccpp-framework
https://github.com/NCAR/ccpp-physics
https://github.com/NCAR/ccpp-doc

CCPP is built to accelerate transition of innovations



What makes a scheme interoperable?

- Well-defined and documented entry points
- Readability of the code (faciliate debugging)
- Expose constants and tuning parameters
- Avoid use of derived or compounded data types
- Standardized error handling, communication

metadata

variable intents

explicit import statements

private/public declarations

use constants and tuning parameters from host model

no assumption on data storage model

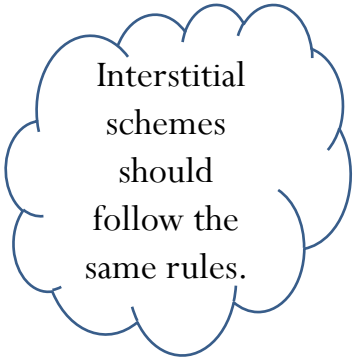
no interference with host model's logging/error handling strategy

no assumptions about parallelization strategy

test and develop with different compilers and optimization flags

What constitutes a CCPP “scheme”?

- Any piece of code with a CCPP-compliant interface:
  - Code must be wrapped within a Fortran module
  - Must contain init, run, and finalize subroutines
  - Must contain CCPP-readable metadata describing argument variables for all subroutines (init/run/finalize)
  - Use CCPP error-tracking variables rather than printing/stopping
  - Have formatted scientific/technical documentation
  - Conform to modern coding standards
- Scheme independence: smallest functional unit possible
  - if scheme functions will always be called together, OK to keep as one
  - if scheme functions will operate independently, separate the schemes



Why are interstitial schemes needed?

- Context
  - In the CCPP world, all physics-related code must be callable using a CCPP-compliant interface
    - Why? Interchangeability, standardization, different suite applicability
  - If we want the GFS physics suite to be used by other models, then code that had been in `GFS_*driver.F90` (which constitutes part of the suite!) must be callable from the CCPP too.
    - If CCPP schemes can contain anything, why aren't `GFS_*driver.F90` considered schemes themselves? This (drastically) reduces flexibility for research.
  - During CCPP origination, DTC staff were responsible for figuring out how to divide the driver code into schemes that made sense (and worked). The current solution is not unique, improvements are welcome.

Guidelines/Process followed

- Useful distinction (evident in current filenames)
  - Can specifically augment a physical parameterization (scheme-specific)
    - additional functionality beyond what scheme code provides, but tied to one specific parameterization
    - decision point: what functionality "belongs" in primary scheme vs interstitial scheme (gray area)
  - Can be suite-level
    - contains functionality that is not tied to one specific scheme but may:
      - provide functionality on top of a class of schemes
      - connects two or more schemes together ("glue code")
      - conversions, initializing sums, applying tendencies

```
…
<scheme>get_phi_fv3</scheme>
<scheme>GFS_suite_interstitial_3</scheme>
<scheme>GFS_DCNV_generic_pre</scheme>
<scheme>samfdeepcnv</scheme>
…
```

Coding standards for portability and readability

- Code must comply to modern Fortran standards (Fortran 90-2008)
  - No `common` blocks, no `goto` statements
  - Prefer `use mpi` instead of `#include mpif.h` (same for OpenMP, NetCDF, …)
- All external information required by the scheme must be passed in via the argument list
  - no `use EXTERNAL_MODULE` for passing in data/constants
- Use explicit import statements, e.g. `use funcphys, only : fpvs`
- Use labeled end statements for modules, subroutines and functions, example:
  - `module scheme_template` → `end module scheme_template`.
- Use `implicit none`.

Coding standards for portability and readability

- All `intent(out)` variables must be set inside the subroutine, including the mandatory variables `errflg` and `errmsg`.
  - Watch out for partially set `intent(out)` variables, these must be `intent(inout)`!
  - Good practice, but also needed: auto-generated code skips variable transformations if the intent information in the metadata indicates that they are not needed
- No permanent state of decomposition-dependent host model data inside the module, i.e. no variables that contain domain-dependent data using the `save` attribute.

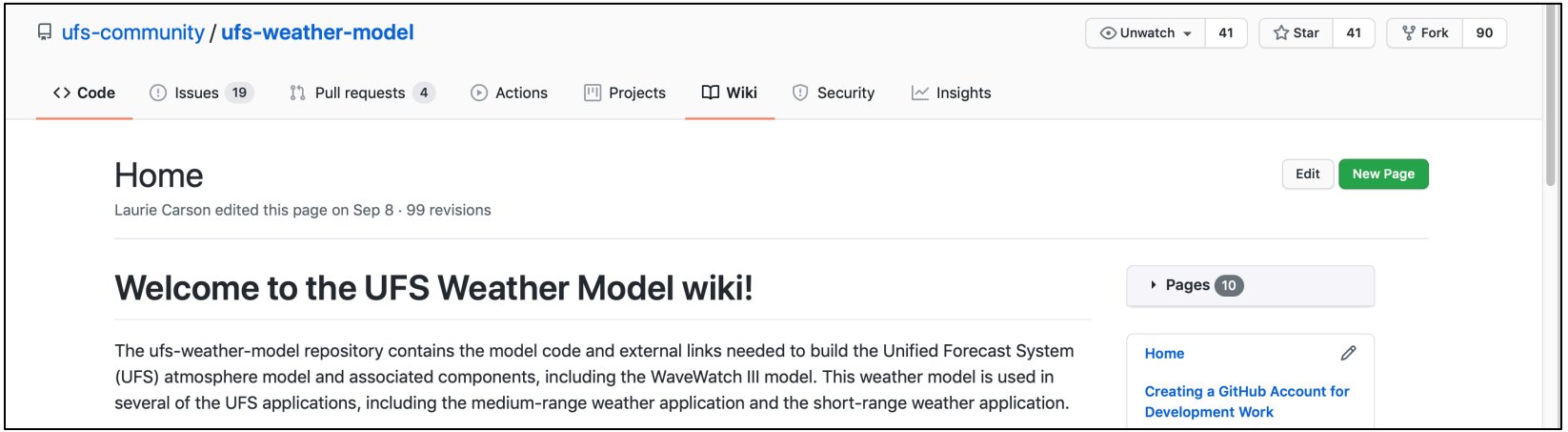
Additional coding rules are listed under the *Coding Standards* section of the NOAA NGGPS Overarching System team document on Code, Data, and Documentation Management for NEMS Modeling Applications and Suites (available at https://docs.google.com/document/u/1/d/1bjnyJpJ7T3XeW3zCnhRLTL5a3m4_3XIAUeThUPWD9Tg/edit#heading=h.97v79689onyd).

Changing gears: using the developmental branch of the ufs-weather-model for contributing code

Switching to ufs-weather-model develop

- The UFS Medium-Range Weather App is not suitable for developmental purposes
  - Code is frozen and lagging the latest development versions from NOAA
  - Workflow does not provide sufficient functionality to support development
  - Unit and regression testing procedures for authoritative repositories use other tools

https://github.com/ufs-community/ufs-weather-model → Wiki



Compiling ufs-weather-model in a nutshell

```
git clone --recursive https://github.com/ufs-community/ufs-weather-model
```

On a supported platform (currently hera.intel, orion.intel, jet.intel, gaea.intel, wcoss_dell_p3, wcoss_cray, cheyenne.intel, cheyenne.gnu):

```
cd ufs-weather-model/tests
./compile.sh MACHINE 'CCPP=Y [SUITES=...]' [LABEL] [CLEAN_BEF.] [CLEAN_AFTER]
```

`SUITES=...` is an optional, comma-separated list of suites (by their suite name) to compile, e.g. `SUITES=FV3_GFS_v15p2,FV3_GFS_v16beta,FV3_GSD_v0`

Using `''` as label gives `fv3.exe` and `modules.fv3` (otherwise `fv3_LABEL.exe`)

Regression testing ufs-weather-model in a nutshell

```
git clone --recursive https://github.com/ufs-community/ufs-weather-model
```

On a supported platform (currently hera.intel, hera.gnu, orion.intel, wcoss_dell_p3, wcoss_cray, cheyenne.intel, cheyenne.gnu):

```
cd ufs-weather-model/tests
export ACCNR=XXX                # project to use for batch jobs
export RT_COMPILER=[intel|gnu] # compiler to use (if supported on machine)
# for intel
./rt.sh -f -e 2>&1 | tee rt_intel.log
# for gnu (where supported)
./rt.sh -l rt_gnu.conf -e 2>&1 | tee rt_gnu.log
```

https://github.com/ufs-community/ufs-weather-model/wiki/Running-regression-test-using-rt.sh

New horizons: adding new schemes

A simple new scheme: qgrs_debug

Suppose some process (scheme) is writing bad data to the 4-dim. tracer array: bad physics? memory leak?

The *old* way:

- Put print statements in `GFS_physics_driver.F90` (and others).
- Use Fortran `.inc` file to avoid making repeated changes.
- Requires searching and editing multiple files, recompiling.

The *new* way:

- Write a simple CCPP scheme with diagnostic output of tracers.
- Add calls to suite definition file where appropriate, recompile.

CCPP scheme qgrs_debug

qgrs_debug.F90

```
module qgrs_debug
...
!! \section arg_table_qgrs_debug_run
!! \htmlinclude qgrs_debug_run.html
!!
subroutine qgrs_debug_run(qgrs,errmsg,errflg)
   real(kind_phys),  intent(in   ) :: qgrs(:,:,:)
   character(len=*), intent(  out) :: errmsg
   integer,          intent(  out) :: errflg
   errmsg = ''
   errflg = 0
   write(0,*) 'qgrs: ', minval(qgrs), maxval(qgrs)
end subroutine qgrs_debug_run
...
end module qgrs_debug
```

CCPP scheme qgrs_debug

qgrs_debug.meta

```
[ccpp-table-properties]
  name = qgrs_debug
  type = scheme
  dependencies = machine.F
[ccpp-arg-table]
  name = qgrs_debug_run
  type = scheme
[qgrs]
  standard_name = tracer_concentration
  long_name = model layer mean tracer concentration
  units = kg kg-1
  dimensions =
(horizontal_loop_extent,vertical_dimension,number_of_tracers)
  type = real
  kind = kind_phys
  intent = in
  optional = F
[errmsg]
...
```

Add scheme to CCPP prebuild

- Edit `FV3/ccpp/config/ccpp_prebuild_config.py`

```
...
SCHEME_FILES = [
    ...
    'ccpp/physics/physics/qgrs_debug.F90',
    ...
    ]
...
```

  - No need to add dependencies, done in scheme metadata
  - No need to add `OPTIONAL_ARGUMENTS` in this case
- Add call to `qgrs_debug` to the suite definition file(s)
- Compile code as usual (recompile w/o clean will do)

Done.

© www.fashionghana.com

Troubleshooting ccpp_prebuild.py

- The CCPP framework code generator `ccpp_prebuild.py` is run automatically as part of `compile.sh` and `rt.sh`.
- If `ccpp_prebuild.py` fails, the compile script/job abort:

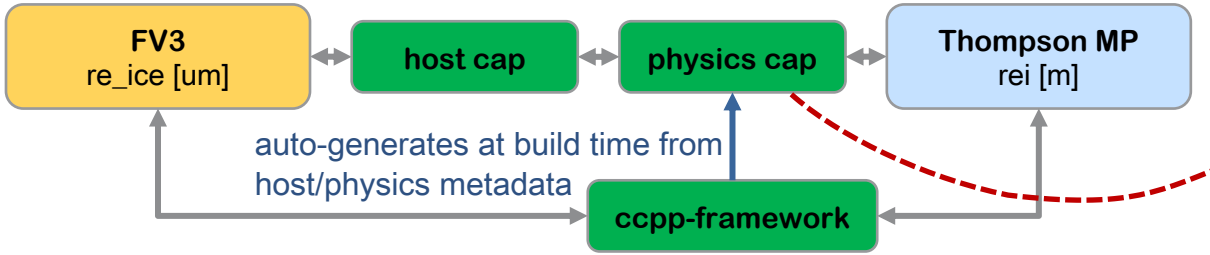
```
An error occured while running ccpp_prebuild.py, check
/Users/…full…path…to…/build_fv3/ccpp_prebuild.{out,err}
```

- Run `ccpp_prebuild.py` manually (since recently from FV3):

```
cd ../FV3/
./ccpp/framework/scripts/ccpp_prebuild.py \
  --config=ccpp/config/ccpp_prebuild_config.py \
  [--suites=...] [--debug] 2>&1 | tee ccpp_prebuild.log
```

Automatic unit conversions

- CCPP supports automatic unit conversions to expedite development and transition.




```
allocate(tmp1, source=re_ice)
tmp1=1.0E-6*re_ice
call thompson_mp(rei=tmp1,...)
re_ice=1.0E+6*tmp1
deallocate(tmp1)
```

- CCPP skips copying data in or out, depending on the intent in the metadata file → beware!
- CCPP knows if an array is allocated or not (`active` attribute) and skips invalid operations.

```
INFO: Automatic unit conversion from C to K for air_temperature before entering
MODULE_mp_thompson SCHEME_mp_thompson SUBROUTINE_mp_thompson_init

Error, automatic unit conversion from K s-1 to C s-1 for
limit_for_temperature_tendency_for_microphysics in MODULE_mp_thompson_post
SCHEME_mp_thompson_post SUBROUTINE_mp_thompson_post_init not implemented
```

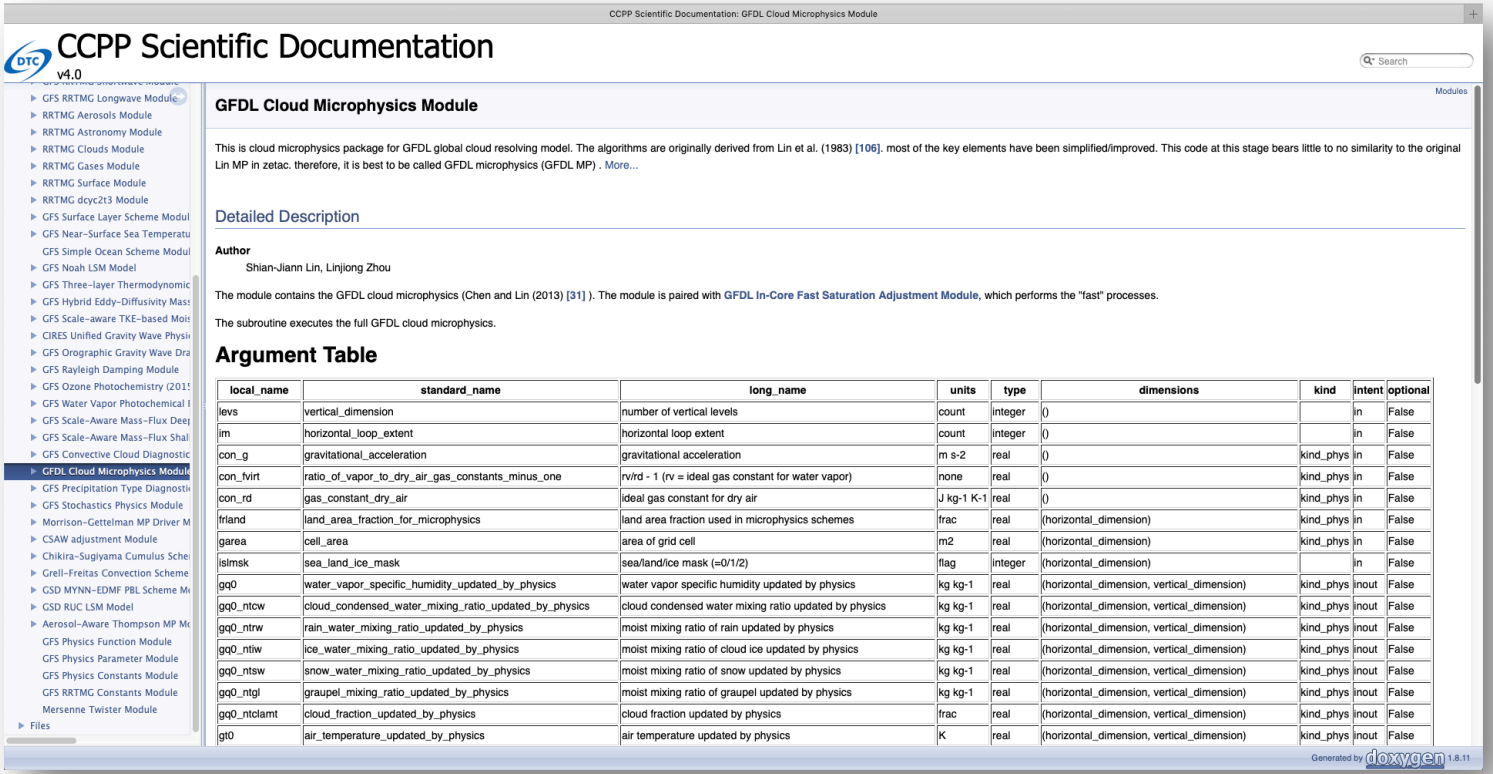
Output from `ccpp_prebuild.py`

Inline documentation using doxygen markup

- CCPP physics rely on inline documentation using doxygen to generate a complete scientific documentation of its parameterizations.
- Documentation of new code/changes in existing code is mandatory for accepting pull requests to the authoritative ccpp-physics repository!
- CCPP metadata is included to generate variable tables for physics schemes

```
!> \file gfdl_cloud_microphys.F90
!! This file contains the CCPP entry point for the column GFDL cloud microphysics
module gfdl_cloud_microphys
...
!>\defgroup gfdlmp  GFDL Cloud Microphysics Module
!! This is cloud microphysics package for GFDL global cloud resolving model.
!! ...
!! \section arg_table_gfdl_cloud_microphys_run Argument Table
!! \htmlinclude gfdl_cloud_microphys_run.html
!!
   subroutine gfdl_cloud_microphys_run(...)
```

Scientific documentation: example GFDL MP



local_name	standard_name	long_name	units	type	dimensions	kind	intent	optional
levs	vertical_dimension	number of vertical levels	count	integer	()		in	False
im	horizontal_loop_extent	horizontal loop extent	count	integer	()		in	False
con_g	gravitational_acceleration	gravitational acceleration	m s-2	real	()	kind_phys	in	False
con_fvirt	ratio_of_vapor_to_dry_air_gas_constants_minus_one	rv/rd - 1 (rv = ideal gas constant for water vapor)	none	real	()	kind_phys	in	False
con_rd	gas_constant_dry_air	ideal gas constant for dry air	J kg-1 K-1	real	()	kind_phys	in	False
frland	land_area_fraction_for_microphysics	land area fraction used in microphysics schemes	frac	real	(horizontal_dimension)	kind_phys	in	False
garea	cell_area	area of grid cell	m2	real	(horizontal_dimension)	kind_phys	in	False
islmsk	sea_land_ice_mask	sea/land/ice mask (=0/1/2)	flag	integer	(horizontal_dimension)		in	False
gq0	water_vapor_specific_humidity_updated_by_physics	water vapor specific humidity updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntcw	cloud_condensed_water_mixing_ratio_updated_by_physics	cloud condensed water mixing ratio updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntrw	rain_water_mixing_ratio_updated_by_physics	moist mixing ratio of rain updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntiw	ice_water_mixing_ratio_updated_by_physics	moist mixing ratio of cloud ice updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntsw	snow_water_mixing_ratio_updated_by_physics	moist mixing ratio of snow updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntgl	graupel_mixing_ratio_updated_by_physics	moist mixing ratio of graupel updated by physics	kg kg-1	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gq0_ntclamt	cloud_fraction_updated_by_physics	cloud fraction updated by physics	frac	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False
gt0	air_temperature_updated_by_physics	air temperature updated by physics	K	real	(horizontal_dimension, vertical_dimension)	kind_phys	inout	False

https://dtcenter.ucar.edu/GMTB/v4.0/sci_doc

Which parameterizations are using variable X?

- Created by `ccpp_prebuild.py` at build time: `CCPP_VARIABLES_FV3.tex`
- Contains a list of all variables used by any of the physics in the SDFs

```
# If using compile.sh as shown before:
cd build_fv3/FV3/ccpp/framework/doc/DevelopersGuide/
# Repeat several times until this message is gone:
# Label(s) may have changed. Rerun to get cross-references right.
pdflatex CCPP_VARIABLES_FV3.tex
# Open CCPP_VARIABLES_FV3.pdf with your preferred PDF viewer
```

- Note: `ccpp_prebuild.py` also creates a list of all variables provided by the host model (independent of choice of SDFs):

```
build_fv3/FV3/ccpp/physics/CCPP_VARIABLES_FV3.html
```

Example output of CCPP_VARIABLES_FV3.pdf

```
air_pressure_difference_between_midlayers
      long_name    air pressure difference between midlayers
      units        Pa
      rank         2
      type         real
      kind         kind_phys
      source       MODULE GFS_typedefs TYPE GFS_interstitial_type
      local_name   GFS_Interstitial(cdata%thrd_no)%del
      requested    GFS_MP_generic_post_run
                   cires_ugwp_run
                   drag_suite_run
                   get_prs_fv3_run
                   gfdl_cloud_microphys_run
                   gwdc_pre_run
                   gwdc_run
                   gwdps_run
                   hedmf_hafs_run
                   hedmf_run
                   moninshoc_run
                   ozphys_2015_run
```

To boldly go where ~~no~~ few developer~~s~~ ha~~s~~ve gone before: modifying or constructing suites

Suite FV3_GFS_v16beta in full glory

```
<?xml version="1.0" encoding="UTF-8"?>

<suite name="FV3_GFS_v16beta" version="1">
  <group name="fast_physics">
    <subcycle loop="1">
      <scheme>fv_sat_adj</scheme>
    </subcycle>
  </group>
  <group name="time
    <subcycle loop
      <scheme>GFS_time_vary_pre</scheme>
      <scheme>GFS_rrtmg_setup</scheme>
      <scheme>GFS_rad_time_vary</scheme>
      <scheme>GFS_phys_time_vary</scheme>
    </subcycle>
  </group>
  <group name="radiation">
    <subcycle loop="1">
      <scheme>GFS_suite_interstitial_rad_reset</scheme>
      <scheme>GFS_rrtmg_pre</scheme>
      <scheme>rrtmg_sw_pre</scheme>
      <scheme>rrtmg_sw</scheme>
      <scheme>rrtmg_sw_post</scheme>
      <scheme>rrtmg_lw_pre</scheme>
      <scheme>rrtmg_lw</scheme>
      <scheme>rrtmg_lw_post</scheme>
      <scheme>GFS_rrtmg_post</scheme>
    </subcycle>
  </group>
  ...
```

```
...
<group name="physics">
    <subcycle loop="1">
      <scheme>GFS_suite_interstitial
      <scheme>GFS_suite_stateout_re
      <scheme>get_prs_fv3</scheme>
                                    _1</scheme>
                                    e</scheme>
                                    _pre</scheme>
      <scheme>GFS_surface_composites_inter</scheme>
      <scheme>GFS_suite_interstitial_2</scheme>
    </subcycle>
    <!-- Surface iteration loop -->
    <subcycle loop="2">
      <scheme>sfc_diff</scheme>
      <scheme>GFS_surface_loop_contro
      <scheme>sfc_nst_pre</scheme>
      <scheme>sfc_nst</scheme>
      <scheme>sfc_nst_post</scheme>
```

```
      ...
      <scheme>GFS_PBL_generic_post</scheme>
      <scheme>GWD_generic_pre</scheme>
      <scheme>_ugwp</scheme>
      <scheme>_ugwp_post</scheme>
      <scheme>GFS_GWD_generic_post</scheme>
      <scheme>rayleigh_damp</scheme>
      <scheme>GFS_suite_stateout_update</scheme>
      <scheme>ozphys_2015</scheme>
      <scheme>h2ophys</scheme>
      <scheme>get_phi_fv3</scheme>
      <scheme>GFS_suite_interstitial_3</scheme>
      <scheme>GFS_DCNV_generic_pre</scheme>
      <scheme>samfdeepcnv</scheme>
      <scheme>GFS_DCNV_generic_post</scheme>
      <scheme>GFS_SCNV_generic_pre</scheme>
      <scheme>samfshalcnv</scheme>
      <scheme>GFS_SCNV_generic_post</scheme>
      <scheme>GFS_suite_interstitial_4</scheme>
      <scheme>cnvc90</scheme>
      <scheme>GFS_MP_generic_pre</scheme>
      <scheme>gfdl_cloud_microphys</scheme>
      <scheme>GFS_MP_generic_post</scheme>
      <scheme>maximum_hourly_diagnostics</scheme>
    </subcycle>
  </group>
  <group name="stochastics">
    <subcycle loop="1">
      <scheme>GFS_stochastics</scheme>
      <scheme>phys_tend</scheme>
    </subcycle>
  </group>
</suite>
```


<suite name="FV3_GFS_v16beta" version="1">

<group name="physics">

<subcycle loop="2">

<scheme>GFS_rrtmg_pre</scheme>
<scheme>rrtmg_sw_pre</scheme>
<scheme>rrtmg_sw</scheme>
<scheme>rrtmg_sw_post</scheme>
<scheme>rrtmg_lw_pre</scheme>
<scheme>rrtmg_lw</scheme>
<scheme>rrtmg_lw_post</scheme>
<scheme>GFS_rrtmg_post</scheme>

Changing or constructing suites

A few things to keep in mind:

- While CCPP allows changing the order of schemes in the suite definition file, this doesn't guarantee that the physics will be correct.
  - Interstitial code needs to be checked and quite often modified
- "Good" interstitial code (modular, depending on one scheme only, with documentation) will facilitate this process, but will make the suite definition file longer.
- `CCPP_VARIABLES_FV3.{tex,pdf,html}` help identifying which schemes read from or write to a variable and which variables are provided by the host model.

Changing or constructing suites (continued)

A few things to keep in mind (continued):

- Changing a suite definition file in many cases requires changing the namelist `input.nml` (must be consistent); also: `field_table`.
- Adding physics in the `fast_physics` or `time_vary` group is more difficult and should be done together with the CCPP developers.
  - `fast_physics` are tightly integrated in dycore & use different variables.
  - `time_vary` called differently than physics/radiation (no blocked data).
  - Work is underway to simplify working with these groups.

Developers **cannot:**

- Add or remove groups in the suite definition file.
  - Exception: group `fast_physics` **must** be removed if `do_sat_adj=.F.`

Example: adding `qgrs_debug` to GFS v16beta

```
<?xml version="1.0" encoding="UTF-8"?>

<suite name="FV3_GFS_v16beta" version="1">
  <group name="fast_physics">
    <subcycle loop="1">
      <scheme>fv_sat_adj</scheme>
      <scheme>GFS_rad_time_vary</scheme>
      <scheme>GFS_phys_time_vary</scheme>
    </subcycle>
  </group>
  <group name="radiation">
    <subcycle loop="1">
      <scheme>GFS_suite_interstitial_rad_reset</scheme>
      <scheme>GFS_rrtmg_pre</scheme>
      <scheme>GFS_rrtmg_post</scheme>
    </subcycle>
  </group>
  ...
```

```
...
<group name="physics">
    <subcycle loop="1">
      <scheme>GFS_suite_interstitial_phys_reset</scheme>
      <scheme>GFS_suite_stateout_reset</scheme>
      <scheme>get_prs_fv3</scheme>
      <scheme>GFS_suite_interstitial_1</scheme>
      <scheme>GFS_surface_generic_pre</scheme>
      <scheme>GFS_surface_composites_pre</scheme>
      <scheme>dcyc2t3</scheme>
      <scheme>GFS_surface_composites_inter</scheme>
      <scheme>GFS_suite_interstitial_2</scheme>
    </subcycle>
    <!-- Surface iteration loop -->
    <subcycle loop="2">
      <scheme>sfc_sice</scheme>
      <scheme>sfc_nst_pre</scheme>
      <scheme>sfc_nst</scheme>
      <scheme>sfc_nst_post</scheme>
      <scheme>sfc_ocean</scheme>
      <scheme>sfc_ocean</scheme>
      <scheme>GFS_surface_loop_control_part2</scheme>
    </subcycle>
    <!-- End surface iteration loop -->
    <scheme>GFS_surface_composites_post</scheme>
    <scheme>sfc_diag</scheme>
    <scheme>sfc_diag_post</scheme>
    <scheme>GFS_surface_generic_post</scheme>
    <scheme>GFS_PBL_generic_pre</scheme>
    <scheme>satmedmfvdifq</scheme>
    ...
```

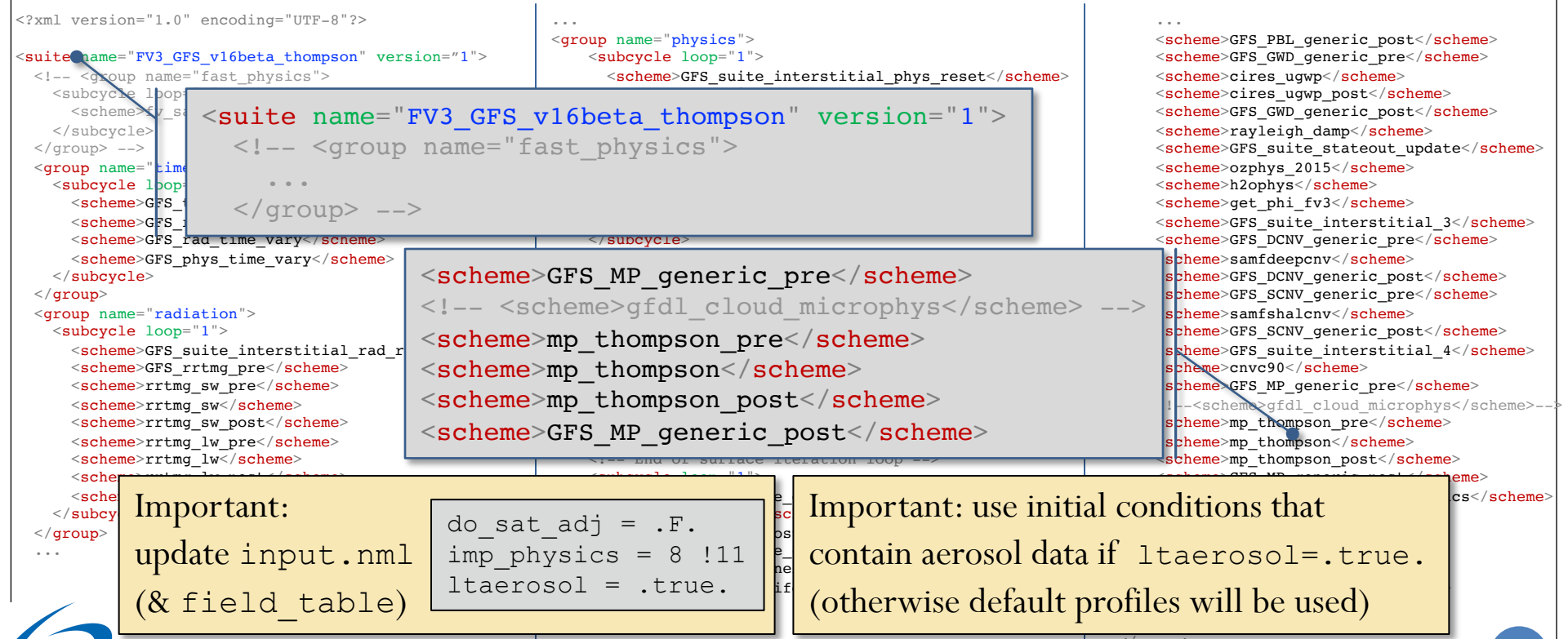
```
      ...
      <scheme>GFS_PBL_generic_post</scheme>
      <scheme>GFS_GWD_generic_pre</scheme>
      <scheme>cires_ugwp</scheme>
      <scheme>cires_ugwp_post</scheme>
      <scheme>GFS_GWD_generic_post</scheme>
      <scheme>rayleigh_damp</scheme>
      <scheme>GFS_suite_stateout_update</scheme>
      <scheme>ozphys_2015</scheme>
      <scheme>h2ophys</scheme>
      <scheme>get_phi_fv3</scheme>
      <scheme>GFS_suite_interstitial_3</scheme>
      <scheme>GFS_DCNV_generic_pre</scheme>
      ...
      <scheme>cnvc90</scheme>
      <scheme>GFS_MP_generic_pre</scheme>
      <scheme>gfdl_cloud_microphys</scheme>
      <scheme>GFS_MP_generic_post</scheme>
      <scheme>maximum_hourly_diagnostics</scheme>
    </subcycle>
  </group>
  <group name="stochastics">
    <subcycle loop="1">
      <scheme>GFS_stochastics</scheme>
      <scheme>phys_tend</scheme>
    </subcycle>
  </group>
</suite>
```

```
<scheme>GFS_suite_stateout_reset</scheme>
<scheme>qgrs_debug</scheme>
<scheme>get_prs_fv3</scheme>
```

```
<scheme>GFS_surface_generic_post</scheme>
<scheme>qgrs_debug</scheme>
<scheme>GFS_PBL_generic_pre</scheme>
```

```
<scheme>GFS_MP_generic_post</scheme>
<scheme>qgrs_debug</scheme>
<scheme>maximum_hourly_diagnostics</scheme>
```

Example: replace GFDL MP with Thompson MP

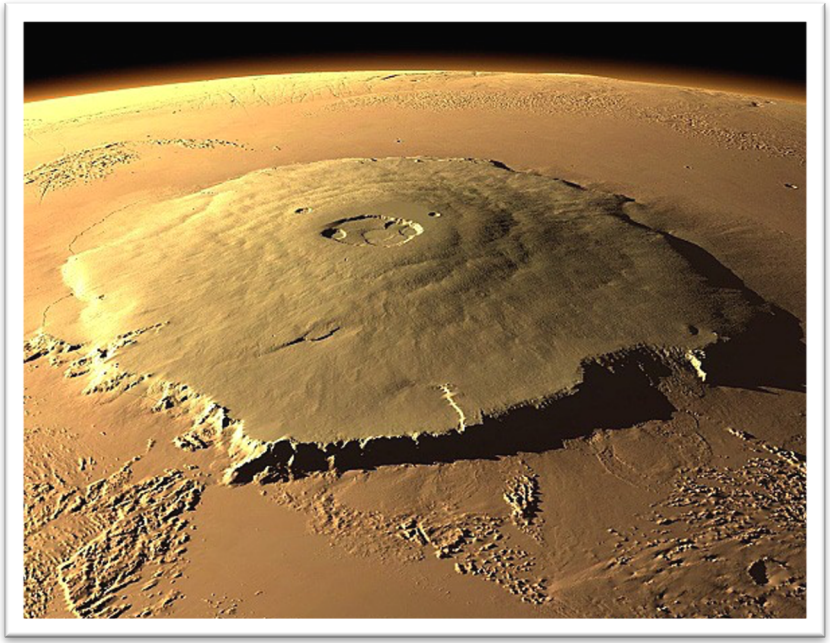


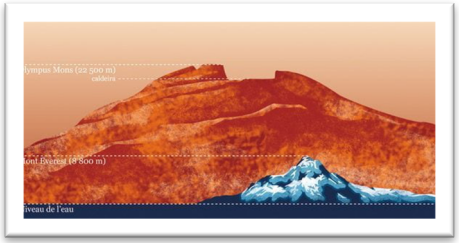
Important:
update `input.nml`
(& `field_table`)

```
do_sat_adj = .F.
imp_physics = 8 !11
ltaerosol = .true.
```

Important: use initial conditions that contain aerosol data if `ltaerosol=.true.` (otherwise default profiles will be used)

Host-side coding





Host-side coding – scenarios

- In many cases, physics developers do not need to change the host model code.
  - Exception: CCPP prebuild config as shown before
- The following scenarios require making host-model changes:
  - An existing variable on the host model side is not yet exposed to the CCPP, i.e. there is no metadata for it.
  - A new variable is required for physics computations.
  - A new or existing variable must be added to the model output (for diagnostics or for restarts).

Adding an existing variable to CCPP – case 1

Case 1. The existing variable is a standard Fortran variable, not a member of a derived data type (constants, flags, …).

- Locate the module in which the variable is defined.
- Add variable to module metadata table if existent.
- If the module doesn't have a metadata table yet:
  - Create metadata table from scratch
  - Add Fortran file to CCPP prebuild config

```
...
VARIABLE_DEFINITION_FILES = [
    ...
    'ccpp/physics/physics/radsw_param.f',
    ...
    ]
...
```

Adding an existing variable to CCPP – case 1

Case 1. The existing variable is a standard Fortran variable, not a member of a derived data type (constants, flags, …).

GFS_typedefs.meta

```
...
[ccpp-arg-table]
  name = GFS_typedefs
  type = module
[LTP]
  standard_name = extra_top_layer
  long_name = extra top layer for radiation
  units = none
  dimensions = ()
  type = integer
...
```

Adding an existing variable to CCPP – case 2

Case 2. The existing variable is a member of a derived data type, (e.g. `GFS_Data(:)%Sfcprop%oro`), the DDT is known to CCPP.

- locate the module in which the DDT is defined
- add variable to the DDT's metadata table

GFS_typedefs.meta

```
[ccpp-arg-table]
  name = GFS_sfcprop_type
  type = ddt
[oro]
  standard_name = orography
  long_name = orography
...
```

Adding an existing variable to CCPP – case 2

Case 2 extra credit. The existing variable is a member of a derived data type, (e.g. `GFS_Data(:)%Sfcprop%tref`), the DDT is known to CCPP. The variable is allocated **conditionally**.

- locate the module in which the DDT is defined, add to DDT metadata table
- translate conditional allocation to `active` attribute in metadata

GFS_typedefs.F90

```
if (Model%nstf_name(1)>0) then
  allocate (Sfcprop%tref(IM))
  ...
end if
```

GFS_typedefs.meta

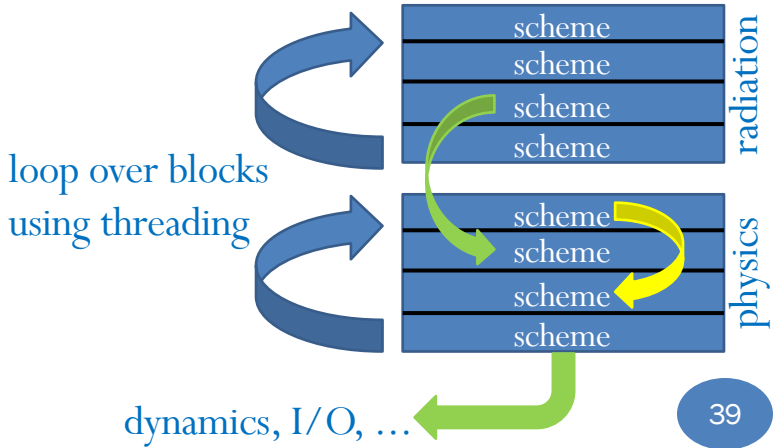
```
[tref]
  standard_name = sea_surface_reference_temperature
  long_name = sea surface reference temperature
  units = K
  dimensions = (horizontal_loop_extent)
  type = real
  kind = kind_phys
  active = (flag_for_nsstm_run > 0)
```

https://ccpp-techdoc.readthedocs.io/en/latest/HostSideCoding.html#active-attribute

Adding a variable to the host+CCPP – case 3

Case 3. The new variable will be a member of a derived data type that already exists on the host model side – this is the easiest case.

- most likely this will be in `GFS_typedefs.F90`
- add constituent array to type definition, allocate and initialize
- reset if applicable (diagnostic or interstitial variables), follow existing code in `GFS_typedefs.F90`
- key question: purpose of this variable
  - interstitial variable: use thread-dependent `GFS_interstitial` DDT
  - persistent variable: use other GFS DDTs (`Diag`, `Sfcprop`, `Tbd`, …)



Adding a variable to the host+CCPP – case 4

Case 4. The new variable is a member of a derived data type that doesn't exist yet in the host model, or the new variable is a derived data type itself that doesn't exist yet in the host model.

- Most developers will not encounter this situation,
  - except if the new variable is a DDT that should become a member of an existing DDT (e.g. `sfcflw_type` in `GFS_radtend_type`).
  - In this case, follow the previous instructions to add a member to a DDT, then add metadata for the new DDT to the existing DDT (see next slide).
  - Other scenarios are not covered here (most complicated cases), contact DTC and UFS code managers/developers if this is really needed.

Adding a variable to the host+CCPP – case 4

radsw_param.meta – defines sfcfsw_type

```
########################################
[ccpp-table-properties]
  name = sfcfsw_type
  type = ddt
  dependencies =

[ccpp-arg-table]
  name = sfcfsw_type
  type = ddt
[upfxc]
  ...
```

GFS_typedefs.meta – defines variable

```
########################################
[ccpp-arg-table]
  name = GFS_radtend_type
  type = ddt
[sfcfsw]
  standard_name = sw_fluxes_sfc
  long_name = sw radiation fluxes at sfc
  units = W m-2
  dimensions = (horizontal_loop_extent)
  type = sfcfsw_type
...
```

Adding a diagnostic variable

- best to use the `GFS_Intdiag_type` in `GFS_typedefs.F90`
- other persistent DDTs will work as well
- follow above instructions for adding a new variable to the DDT
- add code to `GFS_diagnostics.F90` for outputting the data (use an existing entry closest to your needs), for example:

```
idx = idx + 1
ExtDiag(idx)%axes = 2
ExtDiag(idx)%name = 'maxmf'
ExtDiag(idx)%desc = 'maximum mass-flux in column'
ExtDiag(idx)%unit = 'm s-1'
ExtDiag(idx)%mod_name = 'gfs_sfc'
allocate (ExtDiag(idx)%data(nblks))
do nb = 1,nblks
  ExtDiag(idx)%data(nb)%var2 => IntDiag(nb)%maxmf(:)
enddo
```

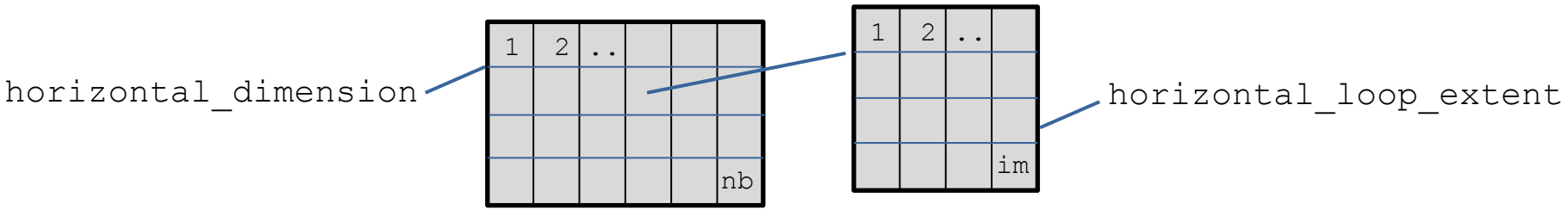
Adding a restart variable

- More complicated than adding a diagnostic variable, because there are several ways to do that:
  - Option 1: add to `GFS_restart_type` in `GFS_restart.F90`
  - Option 2: modify code in `FV3GFS_io.F90`
  - Both require adjusting indices and dimensions, possibly more
- Contact UFS code managers/developers if this is required

horizontal_dimension, horizontal_loop_extent

fv3atm storage model (fv3atm: atmospheric component of ufs-weather-model)

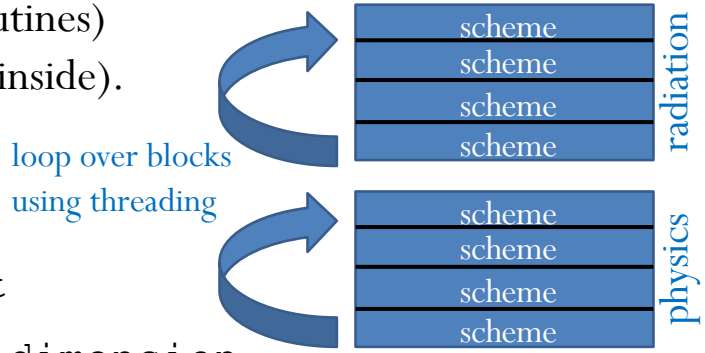
- Example: `GFS_data(1:nb)%Sfcprop%t2m(:)`
- Members of GFS DDTs must use `horizontal_loop_extent`
  - Also includes `GFS_interstitial_type`



horizontal_dimension, horizontal_loop_extent

Support for OpenMP threading in CCPP and implementation in fv3atm

- The `radiation`, `physics` and `stochastics` groups are called for individual blocks, possibly in parallel using several OpenMP threads.
  - This only applies to the time integration (i.e. CCPP `_run` routines).
  - Initialization and finalization (`_init` and `_finalize` routines) are always called for all blocks at once (can use threading inside).
- The `time_vary` and `fast_physics` groups are special and not discussed here.
- CCPP `_run` routines use `horizontal_loop_extent`
- CCPP `_init/_finalize` routines use `horizontal_dimension`



Debugging with CCPP

- Follows the idea discussed above when adding scheme `qgrs_debug`.
- Debugging routines specifically designed for the GFS DDT:

```
FV3/ccpp/physics/physics/GFS_debug.{F90,meta}
```

- `GFS_diagtoscreen`:
  - Print contents of all GFS DDTs except the GFS interstitial DDT
  - min/max/mean or min/max/checksum or individual array elements
  - Configurable by user (change code or CPP directives in `GFS_debug.F90`)
- `GFS_interstitialtoscreen`: as `GFS_diagtoscreen`, but for GFS interstitial
- `GFS_checkland`, `GFS_abort`, … - add your own!

https://ccpp-techdoc.readthedocs.io/en/latest/CCPPDebug.html

Outputting tendencies, auxiliary data in CCPP

- Tendencies for temperature, u/v wind and tracers (qv and ozone) are collected by CCPP per-process (PBL, microphysics, …) and output to disk if requested.
  - Set `ldiag3d=.true.` in `input.nml` to get t, u, v; add `qdiag3d=.true.` for qv, o3
  - Add `<scheme>phys_tend</scheme>` at the end of the stochastics group of the SDF to create sums of physics and non-physics (dycore, nudging, …) tendencies
  - Add the variables of interest to the `diag_table`
  - See also regression test fv3_ccpp_gsd_diag3d_debug for further information
- CCPP supports writing auxiliary 2d/3d data to disk (time average, instantaneous)
  - Set `naux3d`, `naux2d`, `aux3d_time_avg`, `aux_2d_time_avg` in `input.nml`.
  - Pass 2d/3d arrays to scheme like any other variable, write into it what you want.

https://ccpp-techdoc.readthedocs.io/en/latest/ParamSpecificOutput.html

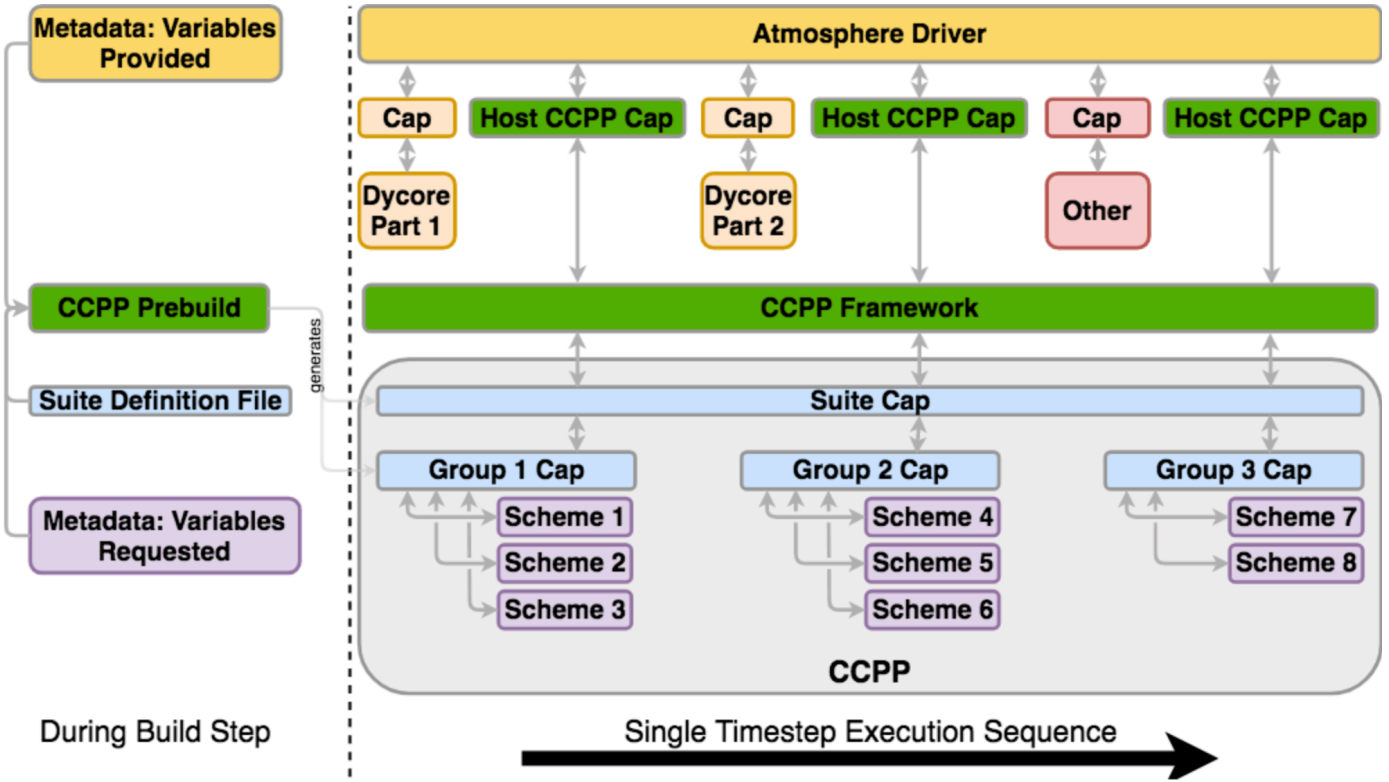
Wrap up

- Writing a new scheme and adding it to a suite is easy.
- Changing or constructing suites may require changing, adding or replacing interstitial code.
- Adding new variables can range anywhere between easy and highly complicated, depending on the situation
  - Defining new standard names should be done in coordination with DTC
- Outputting diagnostic variables is easy; restart variables are more complicated
- CCPP offers tools for debugging, outputting tendencies and auxiliary data

Help is readily available: documentation, forums, tutorials, presentations. Your one-stop shop: https://dtcenter.org/community-code/common-community-physics-package-ccpp

Bonus slides

CCPP Single Timestep Execution Sequence



Parallelization in CCPP: limited MPI, full threading

Overarching paradigms

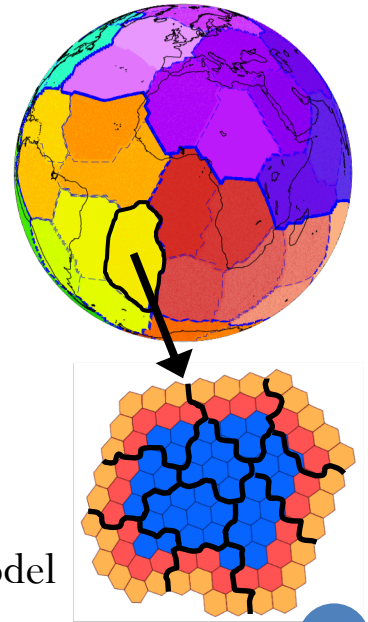
- physics are column-based, no communication during time integration in physics
- physics initialization/finalization can not be called by multiple threads

MPI

- MPI communication only allowed in the physics initialization/finalization
  - need access to entire array of an MPI task (i.e. all blocks) in these phases
- use MPI communicator provided by host model, not `MPI_COMM_WORLD`

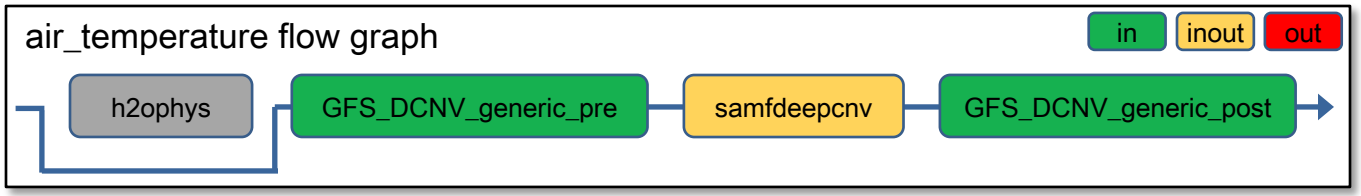
OpenMP

- time integration (but not init./final.) can be called by multiple threads
- threading inside physics is allowed, use # OpenMP threads provided by host model

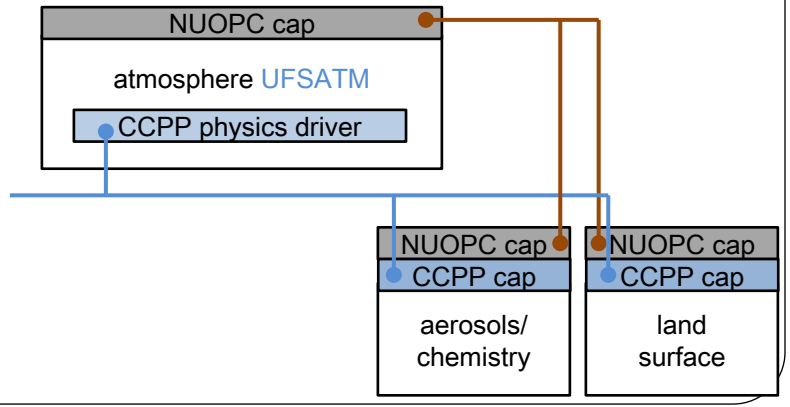


CCPP provides opportunities for future development

- Automatic array transformations: (i,k,j) to (i,k) to (k,i) to …
- Calculation of derived variables: pot. temp. from temp. & geopotential, …



- Automated saving of physics scheme state for restarts
- Extended diagnostic output capabilities from schemes
- Creation of CCPP or NUOPC cap for physics, run either inline or as a separate component (required for UFS) *
- Generation of optimized caps to dispatch physics on CPUs, GPUs, … (required for next-generation HPCs) *



* not funded