

# Common Community Physics Package (CCPP) Overview

Grant Firl<sup>1,2</sup>, Dom Heinzeller<sup>1,3,4</sup>, Ligia Bernardet<sup>1,3</sup>,  
Laurie Carson<sup>1,2</sup>, Man Zhang<sup>1,3,4</sup>, Julie Schramm<sup>1,2</sup>

<sup>1</sup>DTC

<sup>2</sup>NCAR/RAL/JNT

<sup>3</sup>NOAA GSL

<sup>4</sup>CIRES

UFS Training — September 23, 2021

# Outline

- What is the CCPP?
- How does the CCPP fit within a modeling system?
- How are CCPP physics suites defined?
- What makes a piece of code CCPP-compliant?
- How does a host model use the CCPP?
- Further resources

# 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>

**Consolidated:** Single library of operational and developmental parameterizations and suites for all applications

**Supported:** Well-supported community code

**Open:** Have accessible development practices (GitHub)

**Clear interfaces:** Well documented and defined interfaces to facilitate using/enhancing existing parameterizations and adding new parameterizations

**Interoperable:** usable with other dycores/hosts to increase scientific exchange

# What is the CCPP? (1 of 3)

NCAR / **ccpp-physics** Public

<> Code Issues 33 Pull requests 7 Discussion

main 14 branches 12 tags

climbfuji Merge pull request #722 from climbfuji/metadata\_bugfixes

.github	Address Dom's comments, h...	
physics	Merge branch 'main' of https...	
tools	Remove debug print statements from tools/check_encoding.py	16 months ago
.gitignore	Required changes to enable IPD-only, CCPP-only and CCPP-IPD buil...	4 years ago
.gitmodules	Revert change to .gitmodules and update submodule pointer for rte-...	13 months ago
CMakeLists.txt	Update CMakeLists.txt so that local compiler flag modifications also ...	5 months ago
CODEOWNERS	add Julie as codeowner	2 years ago
LICENSE	Add a license file - Apache V2.0 (#50)	4 years ago

- **Library of physical parameterizations**
- **Authoritative fork contains:**
  - **Operational**
  - **Candidates for upcoming implementations**
- **Third-party forks can be used to contain compliant schemes used/developed in other institutions**

# What is the CCPP? (2 of 3)

**NCAR / ccpp-framework** Public

<> Code   Issues 43   Pull requests 1   Discussions

main ▾   8 branches   13 tags

climbujji Merge pull request #391 from climbujji/merge\_feature\_capgen\_i... ✓ 922fe44 26 days ago ⌚ 1,251 commits

.github	Update .github/workflows/python.yaml and tests/test_metadata_par...	last month
cmake	New module to detect OpenMP flags for CCPP build	4 years ago
doc	Merge branch 'feature/capgen' of <a href="https://github.com/NCAR/ccpp-fra...">https://github.com/NCAR/ccpp-fra...</a>	last month
logging	First pass at working ccpp_capgen using version 2 metadata	2 years ago
schema	Removed standard names dictionary, moved to ESCOMP/CCPPStand...	2 years ago
scripts	Remove another legacy block of code in scripts/metavar.py	26 days ago
src	Merge branch 'feature/capgen' of <a href="https://github.com/NCAR/ccpp-fra...">https://github.com/NCAR/ccpp-fra...</a>	last month
test	Fix parsing of dependencies	27 days ago

**Generalized software framework for connecting a set of physical parameterizations with a host application**

- **Model-agnostic**
- **Multi-institutional**

# What is the CCPP? (3 of 3)

NCAR / **ccpp-doc** Public

<> Code Issues Pull requests Actions Projects

main 5 branches 5 tags

climbfuji Merge pull request #38 from SamuelTrahanNOAA

CCPPtechnical	Merge pull request #38 from SamuelTrahanNOAA/feature/more-diag...	2 months ago
CODEOWNERS	Add Ligia to CODEOWNERS	2 years ago
README.md	Fix typo in README.md, change 'Chapter' to 'Section'	2 years ago

☰ README.md ✎

## ccpp-doc

This repository contains the technical documentation for the [GMTB](#) Common Community Physics Package (CCPP). A viewable version of the latest documentation resides [here](#).

## Documentation

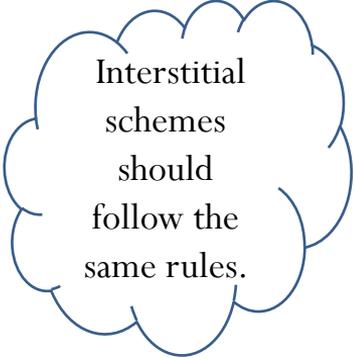
- Physics interfaces
- Framework
- host-model interfaces

## Technical Documentation using ReadTheDocs

- <https://ccpp-techdoc.readthedocs.io/en/v5.0.0/>

# 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



Interstitial schemes should follow the same rules.

# What makes a scheme interoperable?

Well-defined and documented entry points

Readability of the code (facilitate 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

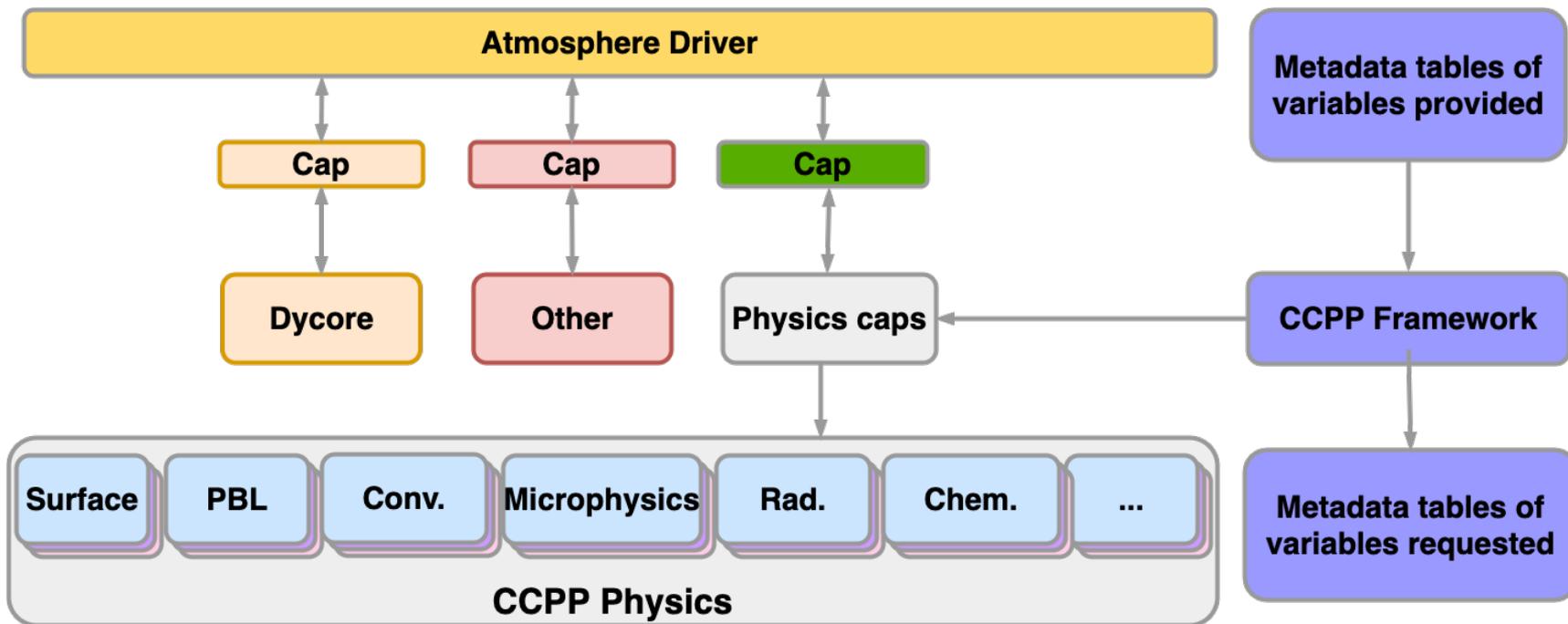
no assumptions about parallelization strategy

test and develop with different compilers and optimization flags

# Primary vs “Interstitial” Schemes

- **Primary Scheme:** a parameterization, such as PBL, microphysics, convection, and radiation, that fits the traditionally-accepted definition.
- **Interstitial Scheme:** a modularized piece of code to perform data preparation, diagnostics, or other “glue” functions that allows primary schemes to work together as a suite.
  - AKA: the code in a traditional physics “driver” between physics scheme calls

# The CCPP Within the Model System



# CCPP Physics Suite Definition

- Individual CCPP-compliant physics parameterizations are assembled and controlled via an XML file called a “**Suite Definition File**” (**SDF**)
- The SDF XML schema has the following hierarchy:
  - Suite
    - Group
      - Subcycle
        - Scheme

Top-level element; defines the suite name and XML schema version

Schemes under one group always get called together in-sequence; non-physics code can be executed between physics groups

Schemes within a subcycle element are executed N times according to the element’s “loop” variable

Each scheme element contains the name of the scheme to run.

# CCPP uses XML suite definition files at build time

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

```
<suite name="FV3_GFS_v16" version="1">
```

```
<group name="fact_physics">
```

```
<subcycle loop="1">
```

```
<scheme>fv3_gfs_physics_pre</scheme>
```

```
</subcycle>
```

```
</group>
```

```
<group name="time_vary">
```

```
<subcycle loop="1">
```

```
<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>GFS_radiation_surface</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>
```

```
...
```

```
<suite name="FV3_GFS_v16" version="1">
```

```
...
```

```
<group name="physics">
```

```
<subcycle loop="1">
```

```
<scheme>GFS_suite_interstitial_1</scheme>
```

```
<scheme>GFS_suite_stateout_res</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_diff</scheme>
```

```
<scheme>GFS_surface_loop_control</scheme>
```

```
<scheme>sfc_nst_pre</scheme>
```

```
<scheme>sfc_nst</scheme>
```

```
<scheme>sfc_nst_post</scheme>
```

```
<scheme>lan_res</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>
```

```
<group name="physics">
```

```
...
```

```
<scheme>GFS_PBL_generic_post</scheme>
```

```
<scheme>GFS_GWD_generic_pre</scheme>
```

```
<scheme>GFS_uugwp</scheme>
```

```
<scheme>GFS_uugwp_post</scheme>
```

```
<scheme>GFS_GWD_generic_post</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>
```

```
<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>
```

# Basic code structure

```
module myscheme
  implicit none

  contains

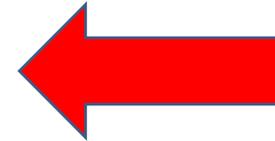
  subroutine myscheme_init ()
  end subroutine myscheme_init

  > \section arg_table_myscheme_run Argument Table
  ! \htmlinclude myscheme_run.html
  !
  subroutine myscheme_run(ni, psfc, errmsg, errflg)
    integer,          intent(in)      :: ni
    real,             intent(inout)   :: psfc(:)
    character(len=*), intent(out)    :: errmsg
    integer,          intent(out)    :: errflg

    ...

  end subroutine myscheme_run

  subroutine myscheme_finalize()
  end subroutine myscheme_finalize
end module myscheme
```



“Hook” for  
CCPP metadata

# CCPP scheme metadata

```
[ccpp-table-properties]
  name = myscheme
  type = scheme
  dependencies = other_file.F90

[ccpp-arg-table]
  name = myscheme_run
  type = scheme

[stress]
  standard_name = surface_wind_stress
  long_name = surface wind stress
  units = m2 s-2
  dimensions = (horizontal_loop_extent)
  type = real
  kind = kind_phys
  intent = in
  optional = F
...
```

**myscheme.meta**

Start of new metadata “table”

name of attached subroutine/module

type = [**scheme**, module,  
DDT, host]

# CCPP scheme metadata

```
[ccpp-table-properties]
  name = myscheme
  type = scheme
  dependencies = other_file.F90

[ccpp-arg-table]
  name = myscheme_run
  type = scheme

[stress]
  standard_name = surface_wind_stress
  long_name = surface wind stress
  units = m2 s-2
  dimensions = (horizontal_loop_extent)
  type = real
  kind = kind_phys
  intent = in
  optional = F
...

myscheme.meta
```

name of variable in  
subroutine

the key by which this data is  
known in the CCPP

more descriptive name if  
standard name is not sufficient

note the format; possibility of  
automatic unit conversion  
among schemes and between  
host

# CCPP scheme metadata

```
[ccpp-table-properties]
  name = myscheme
  type = scheme
  dependencies = other_file.F90

[ccpp-arg-table]
  name = myscheme_run
  type = scheme
[stress]
  standard_name = surface_wind_stress
  long_name = surface wind stress
  units = m2 s-2
  dimensions = (horizontal_loop_extent)
  type = real
  kind = kind_phys
  intent = in
  optional = F
...

myscheme.meta
```

standard names of array dimensions;  
( ) for scalar;  
can specify start:end for dimension  
(default is 1)

FORTRAN intrinsic type or  
DDT name

precision or character length

FORTRAN argument intent

FORTRAN optional argument

# CCPP scheme metadata

```
[ccpp-table-properties]
  name = myscheme
  type = scheme
  dependencies = other_file.F90

[ccpp-arg-table]
  name = myscheme_run
  type = scheme

[stress]
  standard_name = surface_wind_stress
  long_name = surface wind stress
  units = m2 s-2
  dimensions = (horizontal_loop_extent)
  type = real
  kind = kind_phys
  intent = in
  optional = F

...

myscheme.meta
```

Applies to entire scheme;  
dependencies attribute allows  
compiling only those files that  
are necessary for a given list of  
suites

# CCPP error handling

- Schemes should make use of CCPP error-handling variables and not stop/abort/print errors within
- `ccpp_error_flag` and `ccpp_error_message` must be arguments (intent OUT)
- In the event of an error, assign a meaningful error message to **`errmsg`** and set **`errflg`** to a value other than 0:

```
write (errmsg, '(*(a))') 'Logic error in scheme xyz: ...'  
errflg = 1  
return
```

```
[errmsg]  
  standard_name = ccpp_error_message  
  long_name = error message for error  
  ...  
  units = none  
  dimensions = ()  
  type = character  
  kind = len=*  
  intent = out  
  optional = F  
[errflg]  
  standard_name = ccpp_error_flag  
  long_name = error flag for error ...  
  units = flag  
  dimensions = ()  
  type = integer  
  intent = out  
  optional = F
```

# Progress towards a cross-lab standard name resource

The screenshot displays the GitHub interface for the repository `ESCOMP / CCPPStandardNames`. At the top, there's a navigation bar with options like `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, `Insights`, and `Settings`. Below this, the repository name and a brief description are shown. A prominent feature is a merge pull request #10 from `cacraigucar/CAMDEN_sync`, dated Apr 16, with 20 commits. A list of files and their commit messages is provided, such as `.gitignore` (Move standard name database creation code from ccpp-framework) and `write_standard_name_table.py` (Move standard name database creation code from ccpp-framework). The `README.md` content is also visible, explaining the repository's purpose and providing instructions for regenerating the standard name Markdown file using a Python script: `python write_standard_name_table.py standard_names.xml`.

- Active discussion on contents and ways to utilize the repository
- First set of updated standard names and rules for creating new names have been published

# CCPP inline scientific/technical documentation

- Uses Doxygen inline markup
- Additive to existing source code documentation
- Metadata table is parsed into HTML to be included on generated documentation website
- Includes information about scheme provenance, scientific papers, figures, code layout, and scheme algorithm

[https://dtcenter.ucar.edu/GMTB/v5.0.0/sci\\_doc/index.html](https://dtcenter.ucar.edu/GMTB/v5.0.0/sci_doc/index.html)

<https://ccpp-techdoc.readthedocs.io/en/v5.0.0/>

# CCPP coding miscellany

- All external information required by the scheme must be passed in via the argument list.
  - No 'use EXTERNAL\_MODULE' for passing in data
  - Physical constants should go through the argument list
- Code must comply to modern Fortran standards (Fortran 90/95/2003/2008).
- Use labeled **end** statements for modules, subroutines and functions, example:
  - **module scheme\_template** → **end module scheme\_template**.
- Use **implicit none**.
- 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.]
- 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.
- No **goto** statements.
- No **common** blocks.

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](https://docs.google.com/document/u/1/d/1bjnyJpJ7T3XeW3zCnhRLTL5a3m4_3XIAUeThUPWD9Tg/edit#heading=h.97v79689onyd)).

# How can a host use the CCPP?

- See Chapter 6 in the CCPP Documentation:
  - <https://ccpp-techdoc.readthedocs.io/en/v5.0.0/HostSideCoding.html>
- Host metadata (which variables it can provide to physics)
- Calls within code
- Parallelism
- CCPP at build-time
  - Multi-suite compilation
  - What is produced?

# CCPP Host metadata

- Most of the host metadata is in `FV3/ccpp/data/GFS_typedefs.meta`
- Other files also have metadata to help define DDTs or provide other variables to the physics (e.g. `machine.F`)
- Differences compared to scheme metadata:
  - Uses `type = DDT or module`
  - Optional and intent metadata attributes are not used
  - Variables can have `active` attribute:
    - `active = logical expression`
    - Since host models may conditionally allocate memory, the logical expression uses CCPP standard names and represents when the given variable is allocated for use in physics:
      - e.g., `active = (flag_diagnostics_3D)`

# Parallelism using the CCpp

## Overarching paradigms

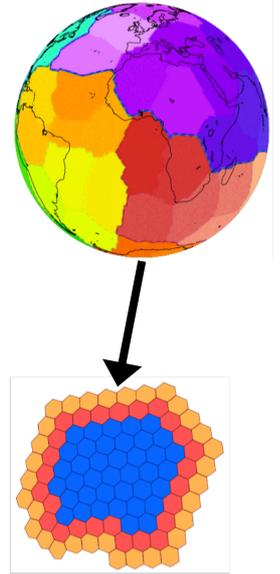
- Physics are column-based, no communication during time integration in physics
- Physics initialization/finalization are independent of threading strategy of the model

## MPI

- MPI communication only allowed in the physics initialization/finalization
- 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 @ build time

- A Python script is the “workhorse” of the CCPP framework and is called at build-time
- The script is given a set of SDFs representing the suites to be compiled and those available to use at run-time
  - Reads all scheme metadata for each given suite
  - Reads all host metadata
  - Matches **variables provided** with **variables requested**
  - Autogenerates suite and group caps
  - Autogenerates `ccpp_static_api.F90`
  - Autogenerates makefile information for compiling physics and caps within host’s build system

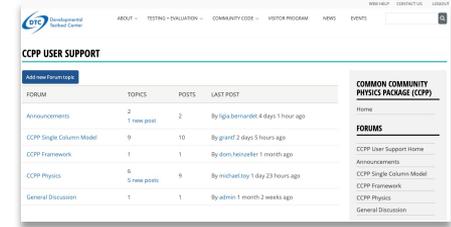
# Future Direction

- Continue to expand contributions and partner with other organizations
- CCPP-physics
  - Continue adding and improving existing schemes to improve UFS applications (e.g. chemistry schemes from NOAA GSL)
- CCPP-framework
  - Transition to new cap generation software (capgen.py; in coordination with NCAR)
  - Usability improvements (e.g. in-suite variable tracking)
  - NUOPC interface for CCPP suites (unfunded)

# Other CCPP support/training resources

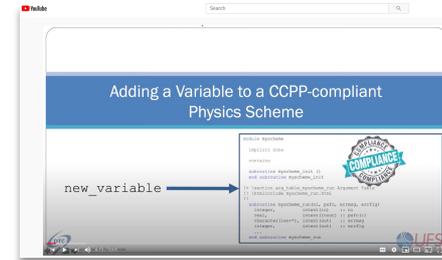
- Forums

- <https://dtcenter.org/forum/ccpp-user-support>
- <https://forums.ufscommunity.org/>



- YouTube

- Developmental Testbed Center Channel
- CCPP playlist
- [https://www.youtube.com/watch?v=ut1mfK5K84w&list=PLFqIc1m9FLQxCpogp6x\\_KQMYvY0BBqY2c](https://www.youtube.com/watch?v=ut1mfK5K84w&list=PLFqIc1m9FLQxCpogp6x_KQMYvY0BBqY2c)



- CCPP Technical Documentation

- <https://ccpp-techdoc.readthedocs.io/en/v5.0.0/>

- CCPP Physics Scientific Docs

- [https://dtcenter.ucar.edu/GMTB/v5.0.0/sci\\_doc/index.html](https://dtcenter.ucar.edu/GMTB/v5.0.0/sci_doc/index.html)

