Common Community Physics Package (CCPP) Overview

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

[1]DTC
[2]NCAR/RAL/JNT
[3]NOAA GSL
[4]CIRES

Outline

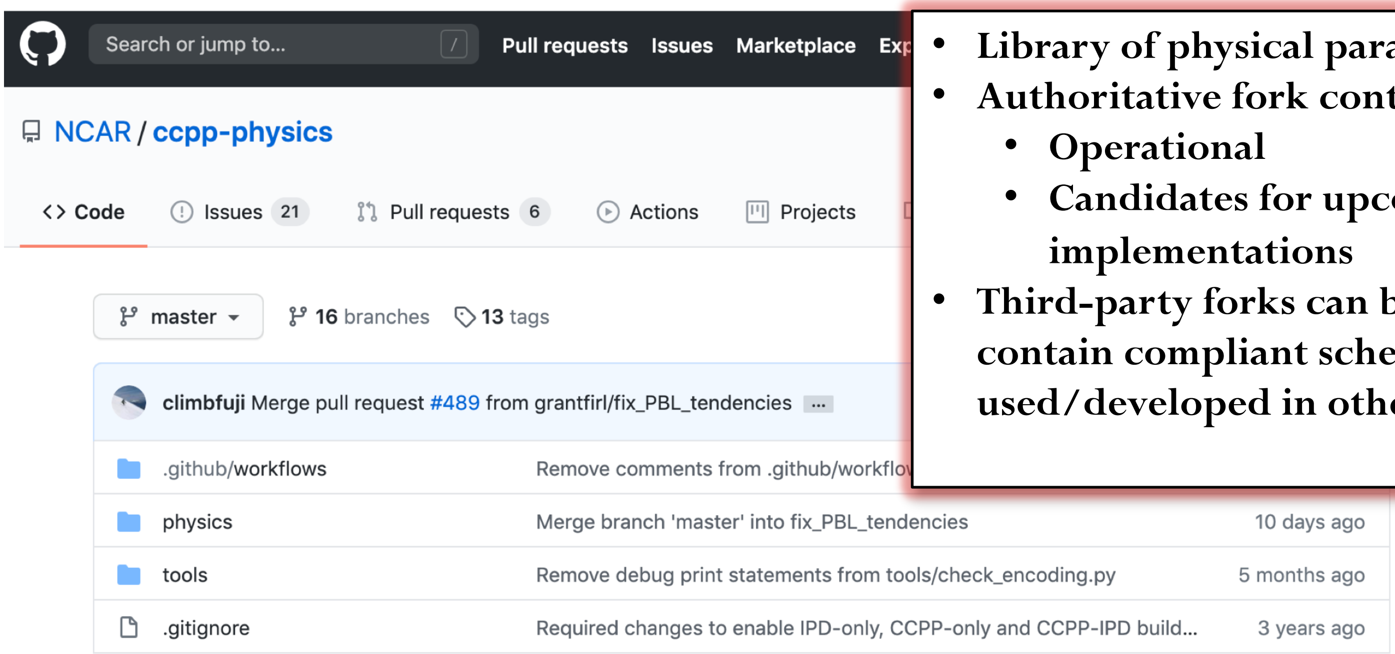
- What is the CCPP?
- How does the CCPP fit within a modeling system?
- How are CCPP physics suites defined?
- What makes a piece of code CCPP-compliant?
- How does a host model use the CCPP?
- What is the history of the CCPP and where is it being used?
- What does the near-term future hold for the CCPP?

Goals for the UFS Physics

- **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
  - Single-Column Model
  - Etc.

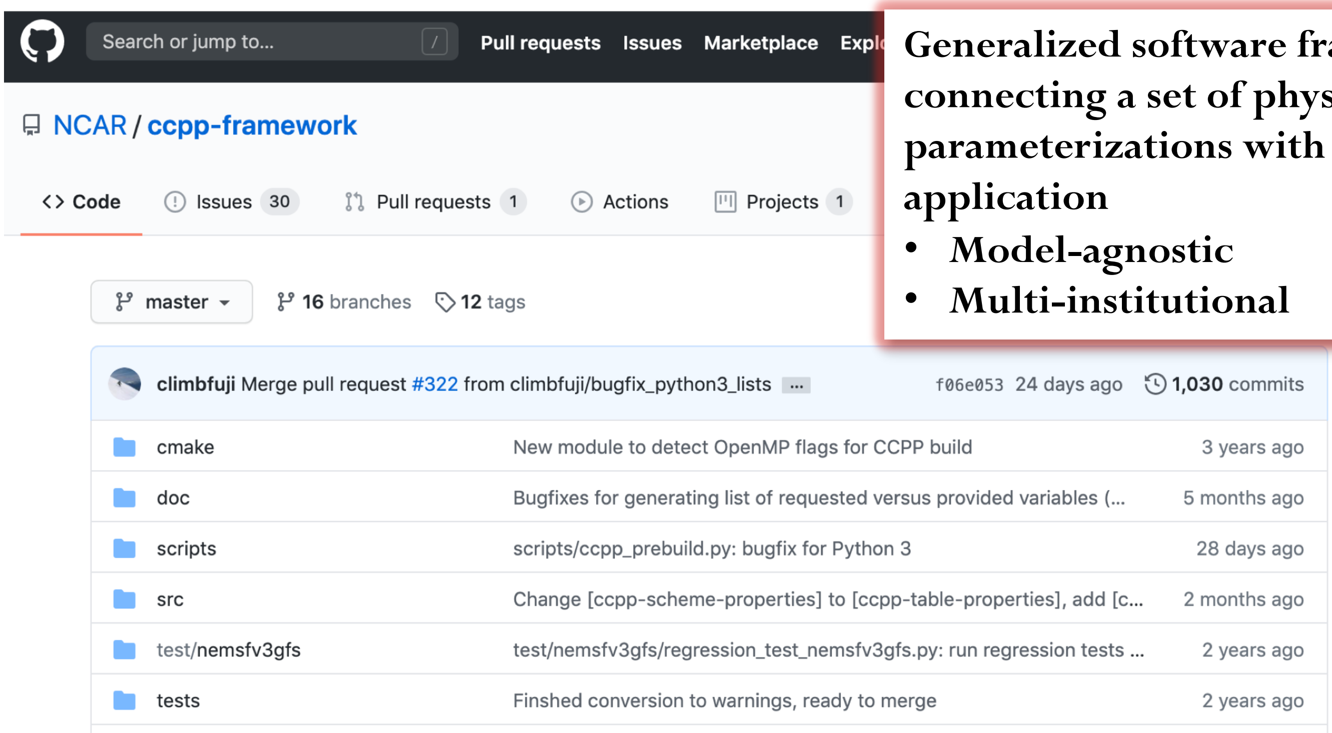
Common Community Physics Package (CCPP)

What is the CCPP? (1 of 2)



- 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

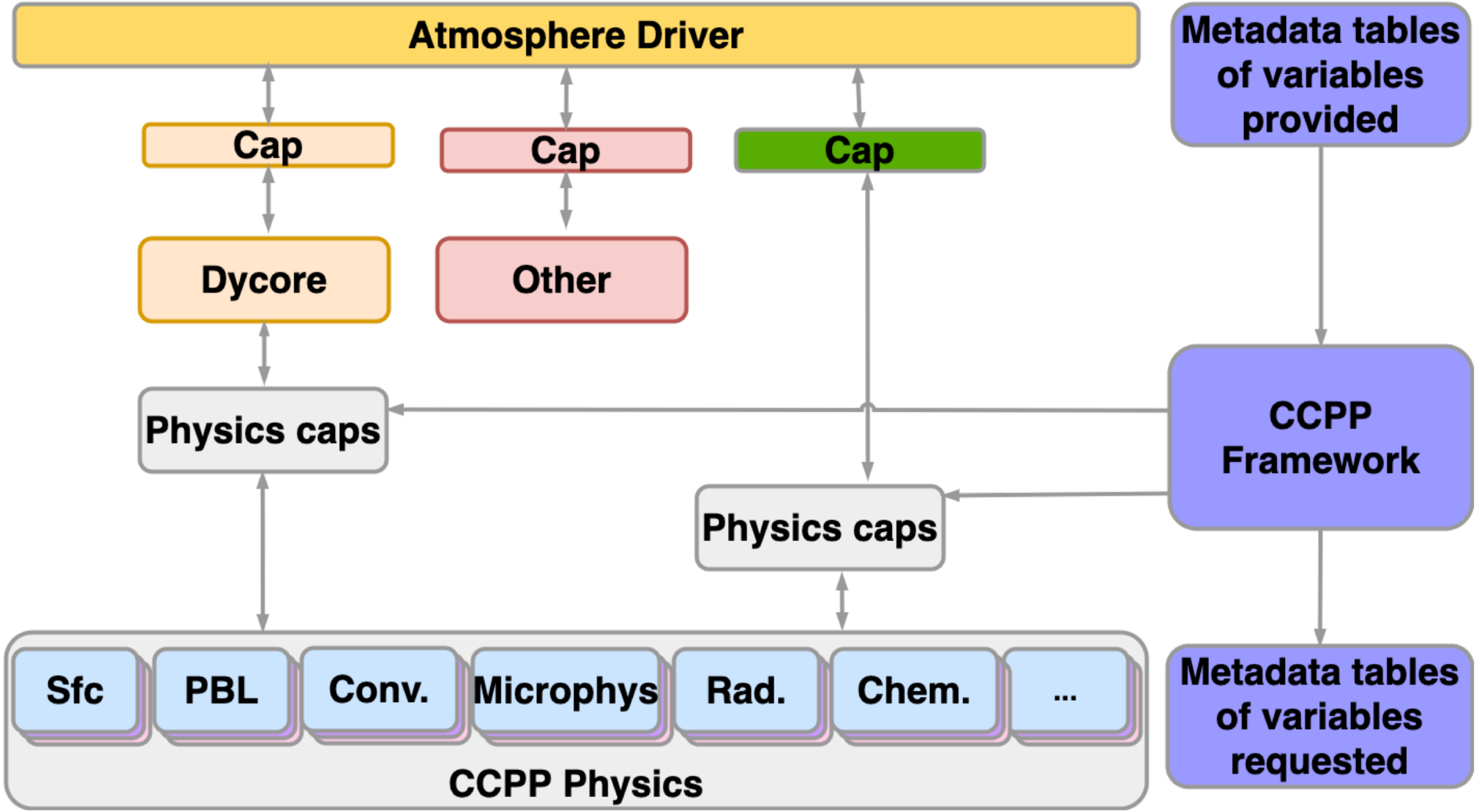
What is the CCPP? (2 of 2)



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

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

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
  "**S**uite **D**efinition **F**ile" (**SDF**)
- The SDF XML schema has the following hierarchy:
  - Suite
    - Group
      - Subcycle
        - Scheme

Element	Description
Suite	Top-level element; defines the suite name and XML schema version
Group	Schemes under one group always get called together in-sequence; non-physics code can be executed between physics groups
Subcycle	Schemes within a subcycle element are executed N times according to the element's "loop" variable
Scheme	Each scheme element contains the name of the scheme to run.

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

What's "special" about a CCPP scheme?

- The interface!
  1. Contained within FORTRAN module
  2. Special `init`, `timestep_init`, `run`, `timestep_finalize` and `finalize` subroutines
  3. **Metadata to describe all arguments in special subroutines**
  4. Special error-handling
  5. Scientific/technical documentation using Doxygen
  6. Modern coding standards
  7. Ability to track variables through schemes in a suite

Basic code structure

```
module myscheme
  implicit none
  contains

!> \section arg_table_myscheme_init Argument Table
!! \htmlinclude myscheme_init.html
!!
  subroutine myscheme_init (errmsg, errflg)
    character(len=*), intent(out)    :: errmsg
    integer,          intent(out)    :: errflg
    ...
  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
end module myscheme
```

myscheme.F90

"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
...
```

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
...
```

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
...
```

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

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
...
```

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_code` 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:

```
[errmsg]
  standard_name = ccpp_error_message
  long_name = error message for error
…
  units = none
  dimensions = ()
  type = character
  kind = len=*
  intent = out
[errflg]
  standard_name = ccpp_error_code
  long_name = error code for error …
  units = flag
  dimensions = ()
  type = integer
  intent = out
```

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

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

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).

How can a host use the CCPP?

- See Chapter 6 in the CCPP Documentation:
  - https://ccpp-techdoc.readthedocs.io/en/v6.0.0/HostSideCoding.html
- Host metadata (which variables it can provide to physics)
- Calls within code
- Parallelism
- CCPP at build-time
  - Multi-suite compilation (static)
  - What is produced?

CCPP Host metadata

- Most of the host metadata is in
  `FV3/ccpp/data/GFS_typedefs.meta`
  `FV3/ccpp/data/CCPP_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`
  - 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)`

CCPP API calls

☆ Autogenerated in ccpp_static_api.F90

- Physics initialization, running, and finalization
  - `ccpp_physics_init`
    - calls the `init` stage of all schemes in the suite (in SDF order)
  - `ccpp_physics_timestep_init`
    - calls the `timestep_init` stage of of the entire suite at once or just one group
  - `ccpp_physics_run`
    - can call the `run` phase of the entire suite at once or just one group
  - `ccpp_physics_timestep_finalize`
    - calls the `timestep_finalize` stage of entire suite at once or just one group
  - `ccpp_physics_finalize`
    - deallocates memory and/or any other run-once finalization work

Parallelism using the CCPP

Overarching paradigms

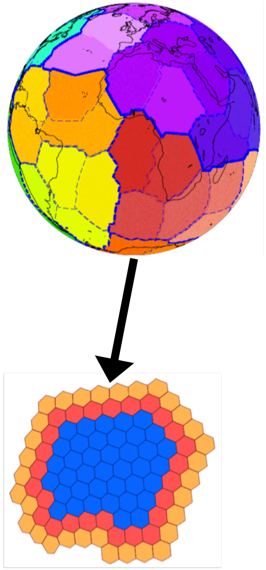
- Physics are column-based, no communication during time integration in physics
- Physics initialization/finalization are independent of threading strategy of the model

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

CCPP History

Ideas
2016

Requirements
2017

Design
2017-ongoing

Develop
2017-ongoing

Public releases
v1-v2-v3:
2018-2019

NCAR/NRL
collaboration
2018-ongoing

Adopted by EMC
2019

Training for NOAA
2019

Public Releases
UFS MRW v1
v4: 2020

UFS Training
2020-2021

Public Releases
UFS SRW v1
v5:2021

Public Releases
UFS SRW v2
v6: 2022

Operations
2023-2024

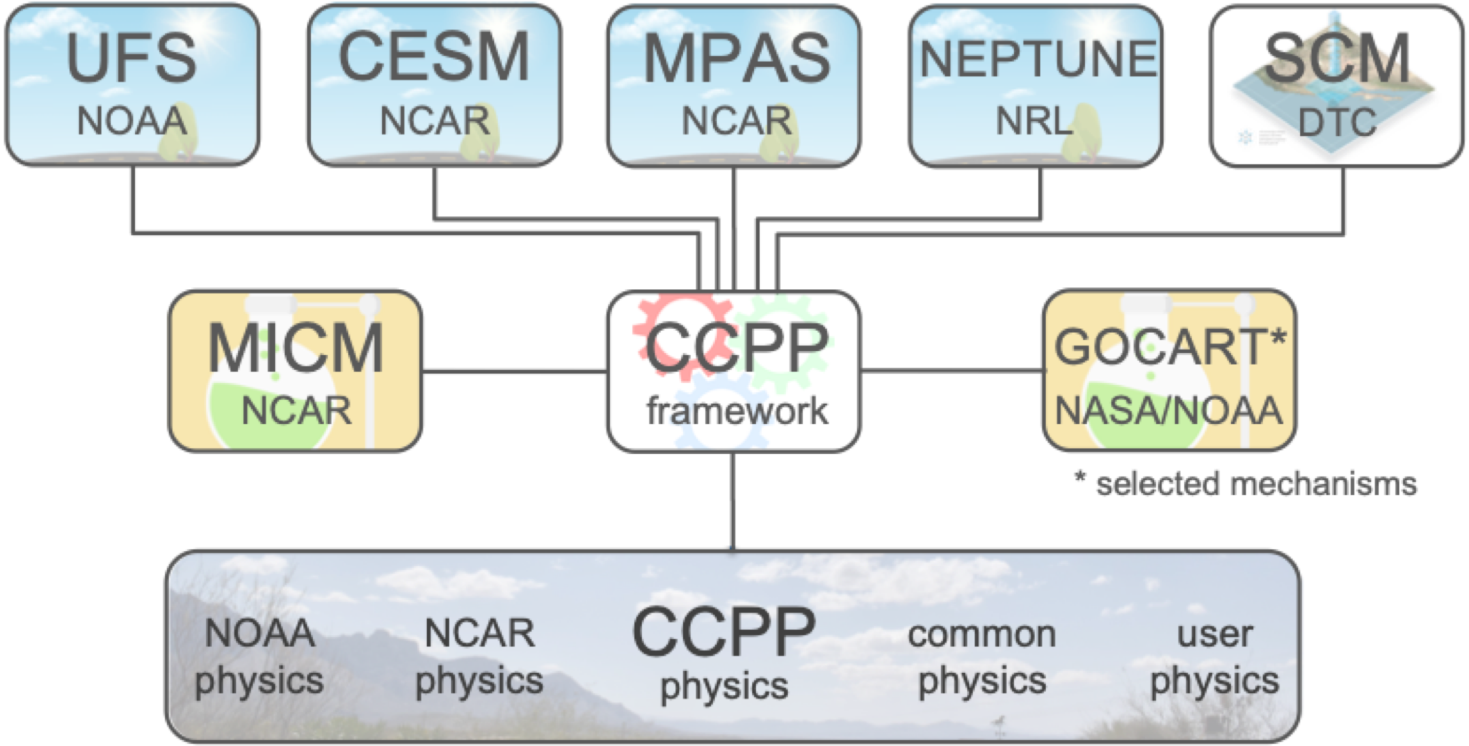
CCPP Public Releases

V	Date	Physics	Host
1.0	2018 Apr	GFS v14 operational	SCM
2.0	2018 Aug	GFS v14 operational updated GFDL microphysics	SCM UFS WM for developers
3.0	2019 Jul	GFS v15 operational Developmental schemes/suites	SCM UFS WM for developers
4.0	2020 Mar	GFS v15 operational Developmental schemes/suites	SCM UFS WM / UFS MRW App v1.0
4.1	2020 Oct	GFS v15 operational Developmental schemes/suites	SCM UFS WM / UFS MRW App v1.1
5.0	2021 Mar	GFS v15 operational Developmental schemes/suites	SCM UFS WM / UFS SRW App v1.0
6.0	2022 Jun	GFS v16 operational Developmental schemes/suites	SCM UFS WM / UFS SRW App v2.0

CCPP v6 supported suites

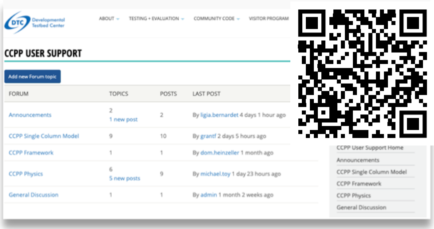
Type	Operational	Developmental				
Suite Name	**GFS_v16**	**GFS_v17_p8**	**HRRR**	**RRFS_v1beta**	**WoFS**	**RAP**
Host	**SRW, SCM**	**SCM**	**SRW, SCM**	**SRW, SCM**	**SRW,SCM**	**SCM**
Microphysics	GFDL	Thompson	Thompson	Thompson	NSSL	Thompson
PBL	TKE EDMF	TKE EDMF	MYNN	MYNN	MYNN	MYNN
Surface Layer	GFS	GFS	MYNN	MYNN	MYNN	GFS
Deep Convection	saSAS	saSAS+CA	—	—	—	Grell-Freitas
Shallow Convection	saMF	saMF	—	—	—	Grell-Freitas
Radiation	RRTMG	RRTMG	RRTMG	RRTMG	RRTMG	RRTMG
Gravity Wave Drag	uGWP	GSL-Drag	GSL-Drag	uGWP	uGWP	GSL-Drag
Land Surface	Noah	Noah-MP	RUC	Noah-MP	Noah-MP	RUC
Ozone	NRL 2015	NRL 2015	NRL 2015	NRL 2015	NRL 2015	NRL 2015
H2O	NRL	NRL	NRL	NRL	NRL	NRL
Aerosols	GOCART	GOCART	GOCART	OPAC	OPAC	GOCART

Models using CCPP

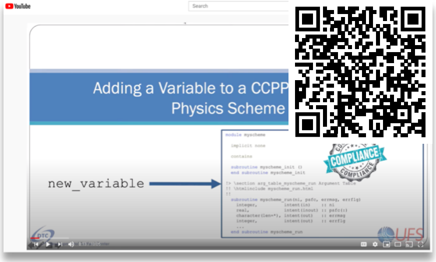


Other CCPP support/training resources

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



- YouTube
  - Developmental Testbed Center Channel
  - CCPP playlist



- CCPP Technical Documentation
  - https://ccpp-techdoc.readthedocs.io/en/v6.0.0/
- CCPP Physics Scientific Docs
  - https://dtcenter.ucar.edu/GMTB/v6.0.0/sci_doc/index.html