CCPP Framework

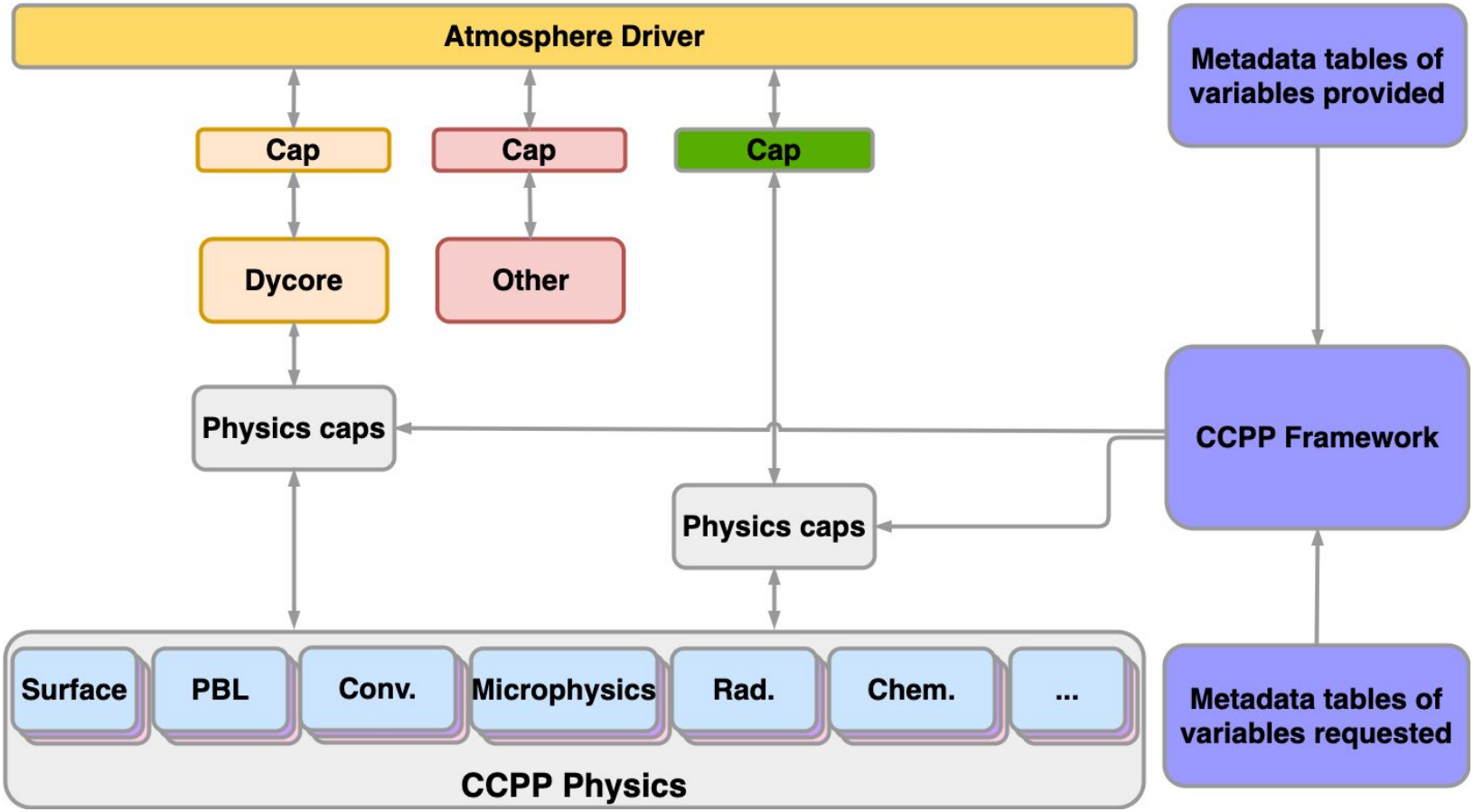
An Overview

Courtney Peverley

Software Engineer - NCAR | CGD

March 4, 2024

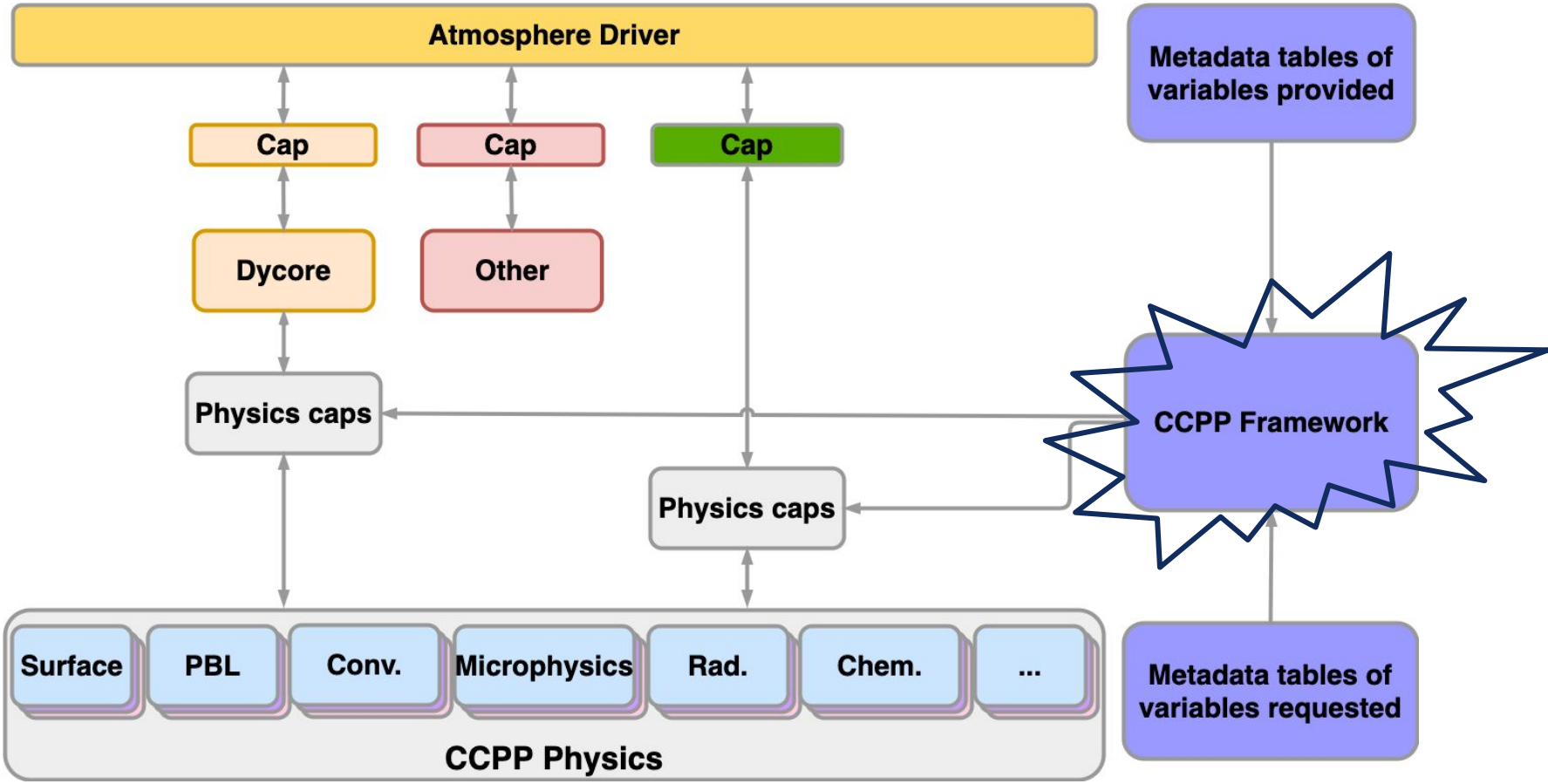
Common Community Physics Package (CCPP)



The CCPP is a software framework that automatically generates the Fortran interface (cap) layer for a physics parameterization (scheme).

*Slide courtesy of Jesse Nusbaumer

Common Community Physics Package (CCPP)

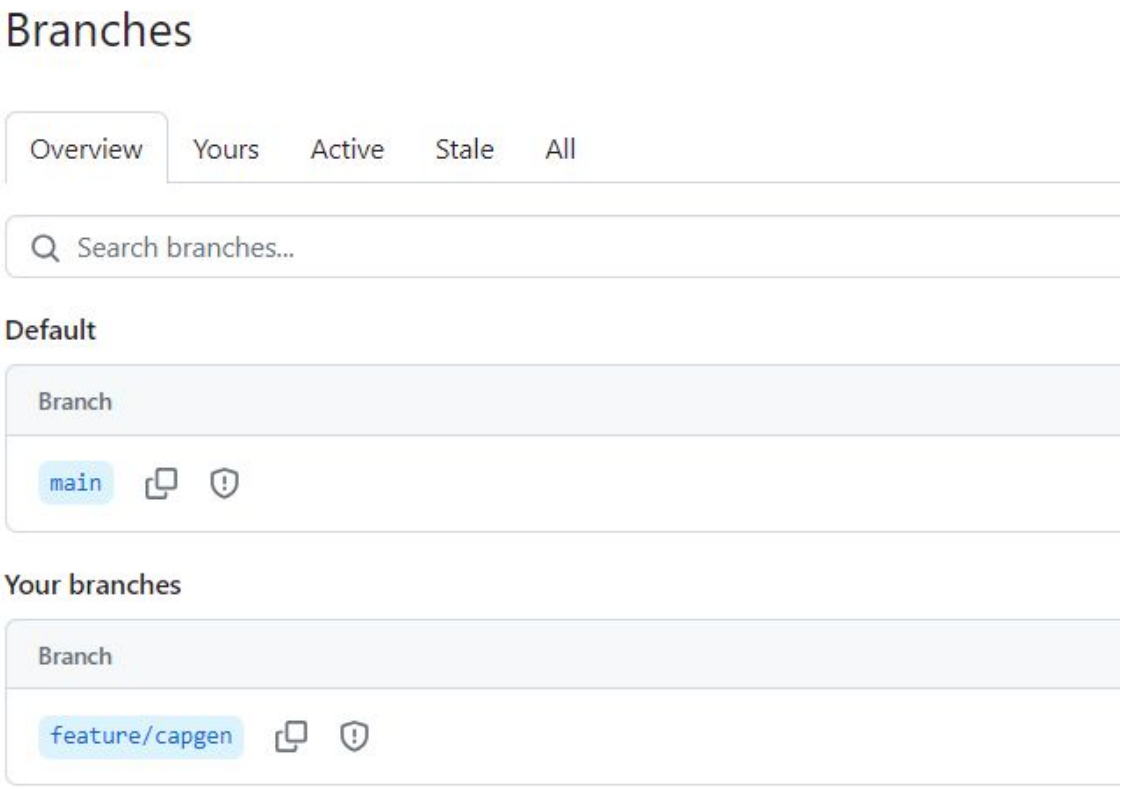


The CCPP is a software framework that automatically generates the Fortran interface (cap) layer for a physics parameterization (scheme).

*Slide courtesy of Jesse Nusbaumer

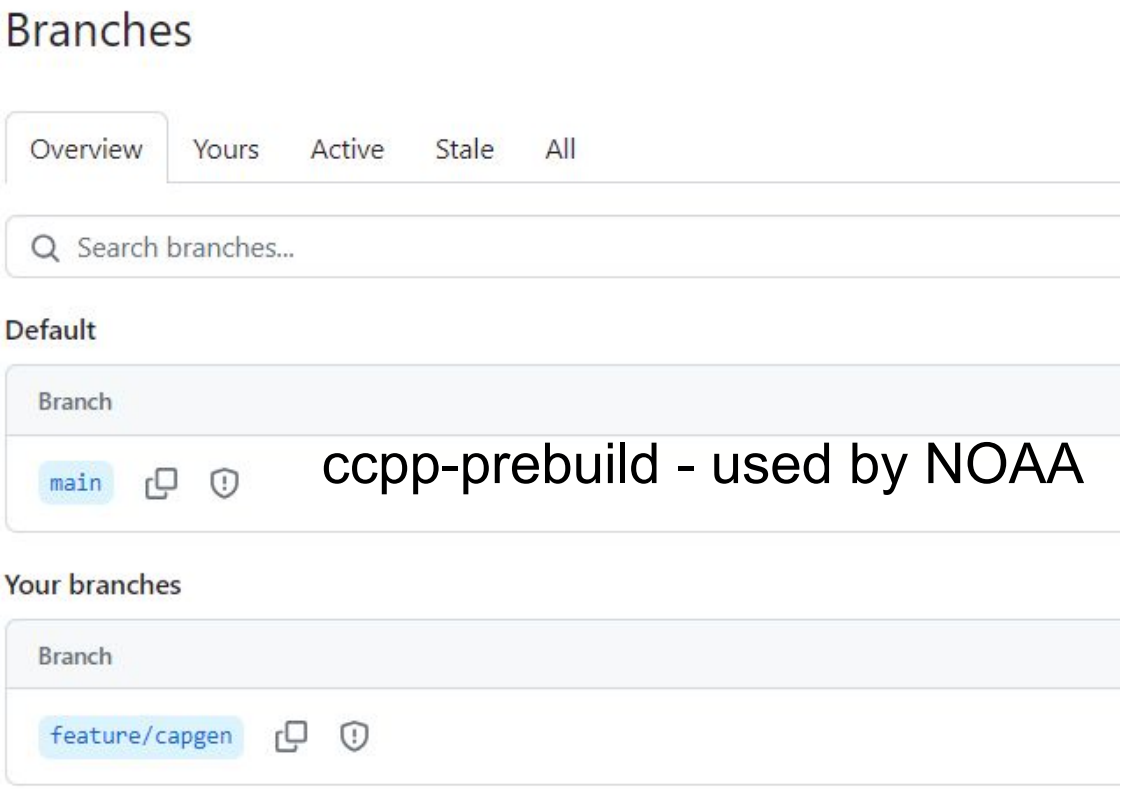
CCPP Framework

Repository: https://github.com/NCAR/ccpp-framework



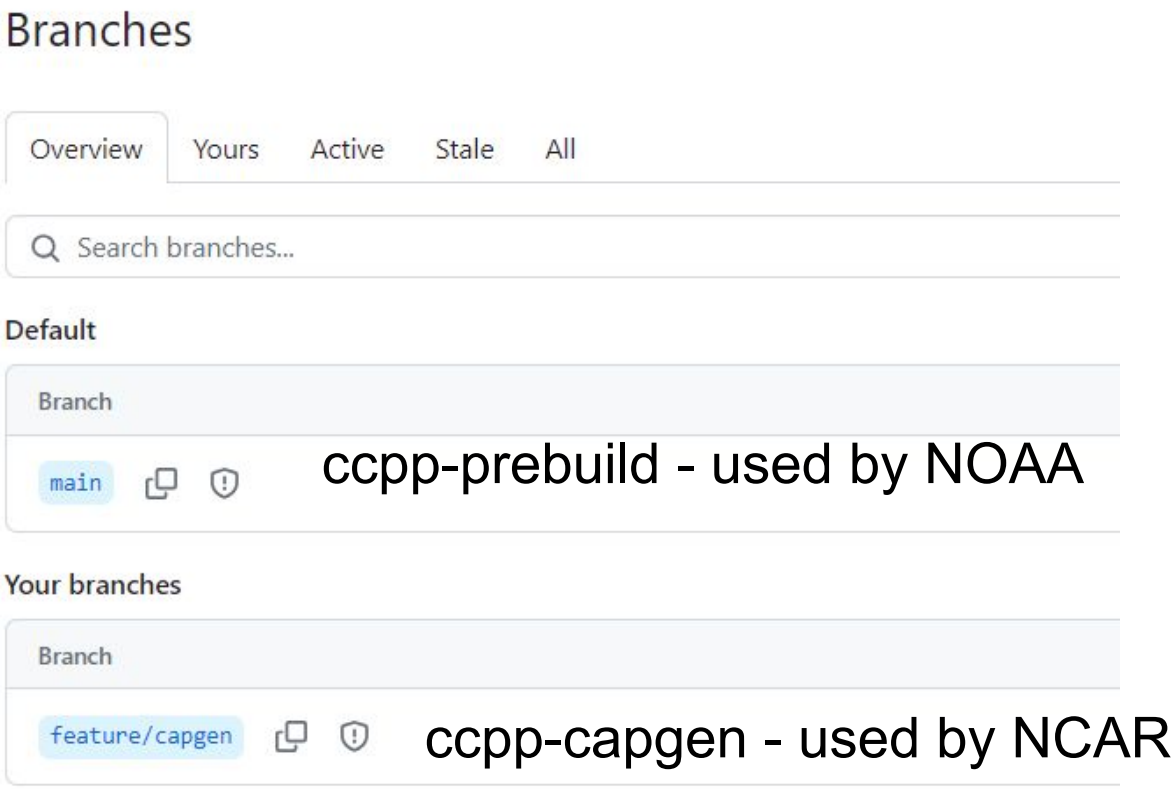
CCPP Framework

Repository: https://github.com/NCAR/ccpp-framework



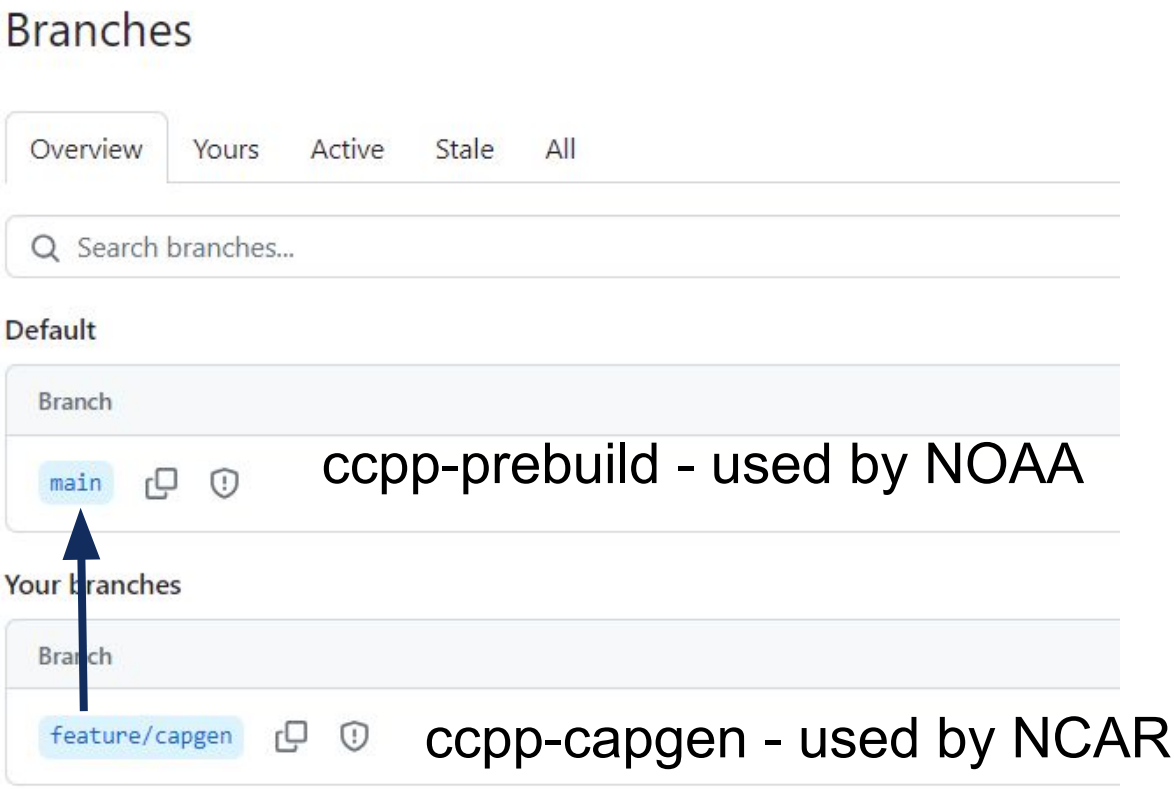
CCPP Framework

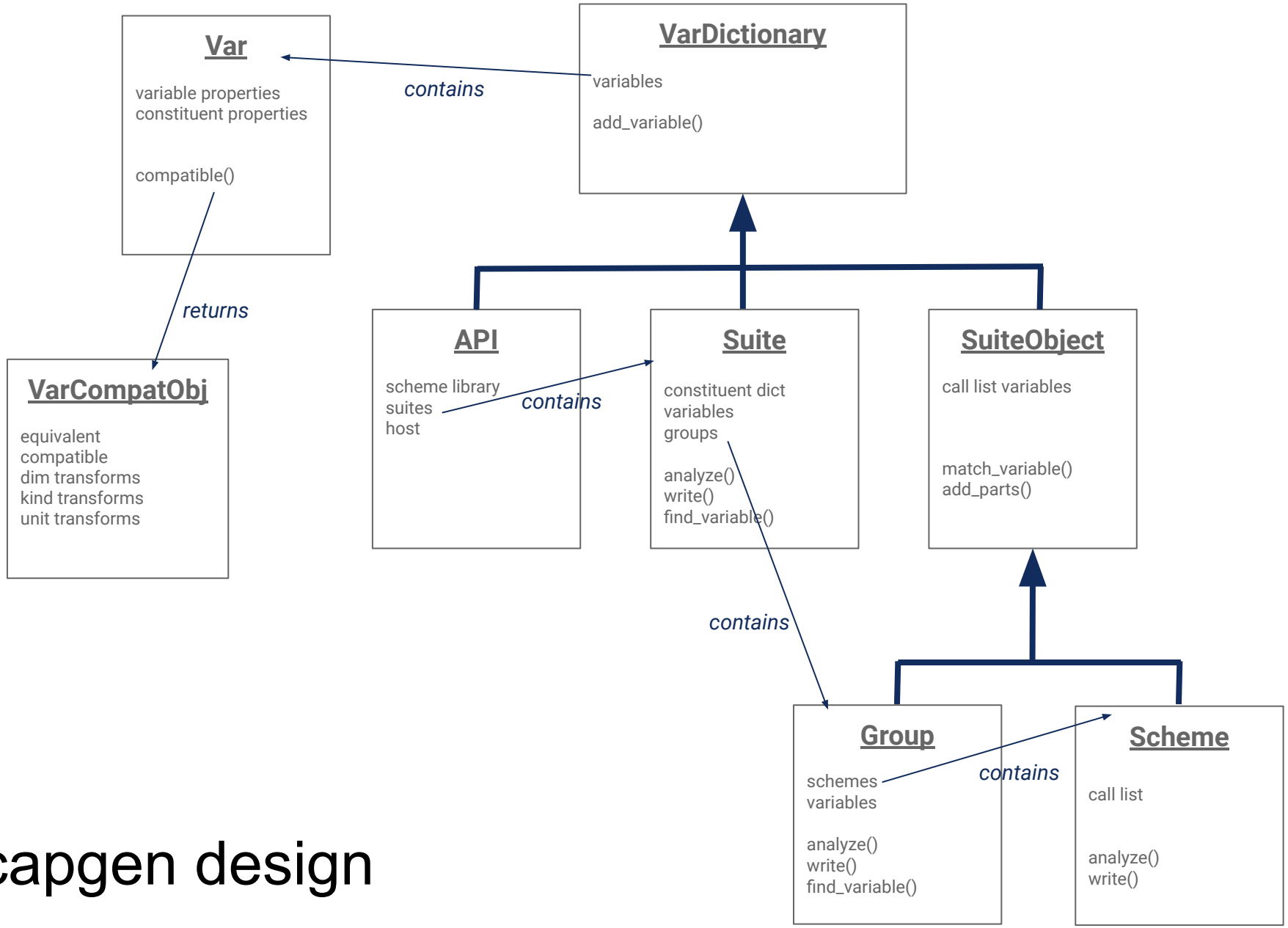
Repository: https://github.com/NCAR/ccpp-framework



CCPP Framework

Repository: https://github.com/NCAR/ccpp-framework





capgen design

Generated code

```xml
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
public :: mysuite_timestep_final
public :: mysuite_finalize
```

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host model calls these caps

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
public :: mysuite_timestep_final
public :: mysuite_finalize
```

Generated code

```xml
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
public :: mysuite_timestep_final
public :: mysuite_finalize
```

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
```

will call courtney_init

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
```

Will do nothing

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
```

Will call courtney_run

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
```

Will call peverley_run

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
public :: mysuite_timestep_final
```

Will call peverley_timestep_final

Generated code

```
<suite name="mysuite" version="1.0">
  <group name="physics_before_coupler">
    <scheme>courtney</scheme>
  </group>
  <group name="physics_after_coupler">
    <scheme>peverley</scheme>
  </group>
</suite>
```

```
module courtney

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: courtney_init ! init routine
   public :: courtney_run  ! main routine
```

```
module peverley

   use ccpp_kinds, only:  kind_phys

   implicit none
   private
   save

   public :: peverley_run  ! main routine
   public :: peverley_timestep_final
```

host_ccpp_cap.F90

```
public :: host_ccpp_physics_timestep_initial
public :: host_ccpp_physics_timestep_final
public :: host_ccpp_physics_initialize
public :: host_ccpp_physics_finalize
public :: host_ccpp_physics_run
```

ccpp_mysuite_cap.F90

```
public :: mysuite_initialize
public :: mysuite_timestep_initial
public :: mysuite_physics_before_coupler
public :: mysuite_physics_after_coupler
public :: mysuite_timestep_final
public :: mysuite_finalize
```

Will do nothing

New capgen features - constituents

- metadata attribute(s) tell the framework to treat a variable as a constituent
  - IN PROGRESS: constituents can also be added at runtime by a scheme
- constituents can then be used:
  - passing in constituents array via standard name: ccpp_constituents
  - just using the standard name for the constituent you want
- constituent object contains an array of constituent properties
  - can access via standard name: ccpp_constituent_properties

```
type, public :: ccpp_model_constituents_t
   ! A ccpp_model_constituents_t object holds all the metadata and field
   !   data for a model run's constituents along with data and methods
   !   to initialize and access the data.
   !!XXgoldyXX: To do: allow accessor functions as CCPP local variable
   !!                  names so that members can be private.
   integer                              :: num_layer_vars = 0
   integer                              :: num_advected_vars = 0
   integer,                  private    :: num_layers = 0
   type(ccpp_hash_table_t), private     :: hash_table
   logical,                  private    :: table_locked = .false.
   logical,                  private    :: data_locked = .false.
   ! These fields are public to allow for efficient (i.e., no copying)
   !   usage even though it breaks object independence
   real(kind_phys), allocatable         :: vars_layer(:,:,:)
   real(kind_phys), allocatable         :: vars_minvalue(:)
   ! An array containing all the constituent metadata
   ! Each element contains a pointer to a constituent from the hash table
   type(ccpp_constituent_prop_ptr_t), allocatable :: const_metadata(:)
```

New capgen features - automatic unit conversions

- Dustin Swales implemented the infrastructure to enable unit conversions
- A limited suite of unit conversions are available and will be added to going forward
- Example:

```
[ pref_in ]
  standard_name = reference_pressure
  long_name = reference pressure used in definition of Exner function
  units = Pa
  dimensions = ()
  type = real | kind = kind_phys
  intent = in
```

```
[ pref ]
  standard_name = reference_pressure
  units = hPa
  type = real | kind = kind_phys
  dimensions = ()
  protected = True
```

```
if (errflg == 0) then
   ! Compute reverse (pre-scheme) transforms
   pref_in_local = 1.0E+2_kind_phys*pref_in

   ! Call scheme
   call kessler_init(lv_in=lv_in, pref_in=pref_in_local, rhoqr_in=rhoqr_in, errmsg=errmsg,  &
        errflg=errflg)


end if
```

Any Questions?

(thank you!)