Recent CLM4.5 Refactoring

>>Terms of Use
>>Go BACK to previous page.

We have recently completed a number of significant code refactorings in the CLM 4_5 code base. You will only notice differences in the code if your starting point is prior to clm4_5_10 and you are merging to a recent tag. We strongly urge you to look over the source code in a CLM trunk tag that is at least as recent as clm4_5_36. Please feel free to contact us at CLM-CMT@cgd.ucar.edu with any questions and we will be happy to assist you.

Parameters in CLM 4_5

- We are in the process of moving parameters and physical constants out of assignments in code to values stored in netcdf files. This makes our code easier to maintain and modify, but most importantly, allows users to do sensitivity analysis tests in an easier way. There are three points to our approach that are worth explaining by example:
    1. If a parameter is only used within one module, then that parameter is private to that module and read by a routine that is a part of that module.

Private Parameters - example from CNAllocationMod.F90

```
type :: CNAllocParamsType
   real(r8) :: bdnr              !bulk denitrification rate (1/s)
   real(r8) :: dayscrecover      !number of days to recover negative cpool
   real(r8) :: compet_plant_no3  ! (unitless) relative compettiveness of plants for NO3
   real(r8) :: compet_plant_nh4  ! (unitless) relative compettiveness of plants for NH4
   real(r8) :: compet_decomp_no3 ! (unitless) relative competitiveness of immobilizers for NO3
   real(r8) :: compet_decomp_nh4 ! (unitless) relative competitiveness of immobilizers for NH4
   real(r8) :: compet_denit      ! (unitless) relative competitiveness of denitrifiers for NO3
   real(r8) :: compet_nit        ! (unitless) relative competitiveness of nitrifiers for NH4
end type CNAllocParamsType
...
type(CNAllocParamsType),protected ::  CNAllocParamsInst
...
subroutine readCNAllocParams ( ncid )
...
  ! read in parameters
  tString='bdnr'
  call ncd_io(varname=trim(tString),data=tempr, flag='read', ncid=ncid, readvar=readv)
  if ( .not. readv ) call endrun( trim(subname)//trim(errCode)//trim(tString))
  CNAllocParamsInst%bdnr=tempr

  tString='dayscrecover'
  call ncd_io(varname=trim(tString),data=tempr, flag='read', ncid=ncid, readvar=readv)
  if ( .not. readv ) call endrun( trim(subname)//trim(errCode)//trim(tString))
  CNAllocParamsInst%dayscrecover=tempr
...
end subroutine readCNAllocParams
!
! now use the parameters in CNAllocationInit
!
bdnr         = CNAllocParamsInst%bdnr * (dt/secspday)
dayscrecover = CNAllocParamsInst%dayscrecover
```

    2. If a parameter is shared by more than one module, then it is placed in a module whose only purpose is to read that shared module.

Shared Parameters - example from CNSharedParamsMod.F90 and CNDecompCascadeCNMod.F90

```
type, public  :: CNParamsShareType
   real(r8) :: Q10         ! temperature dependence
   real(r8) :: minpsi      ! minimum soil water potential for heterotrophic resp
   real(r8) :: cwd_fcel    ! cellulose fraction of coarse woody debris
   real(r8) :: cwd_flig    ! lignin fraction of coarse woody debris
   real(r8) :: froz_q10    ! separate q10 for frozen soil respiration rates
   real(r8) :: decomp_depth_efolding ! e-folding depth for reduction in decomposition (m)
   real(r8) :: mino2lim    ! minimum anaerobic decomposition rate as a fraction of potential
aerobic rate
   real(r8) :: organic_max ! organic matter content (kg/m3) where soil is assumed to act like peat
end type CNParamsShareType
...
type(CNParamsShareType),protected :: CNParamsShareInst
!
! read the shared parameters
!
subroutine CNParamsReadShared(ncid)
...
   tString='mino2lim'
   call ncd_io(trim(tString),tempr, 'read', ncid, readvar=readv)
   if ( .not. readv ) call endrun( trim(subname)//trim(errCode)//trim(tString))
   CNParamsShareInst%mino2lim=tempr
...
  end subroutine CNParamsReadShared
!
! then use in CNDecompCascadeCNMod
!
use CNSharedParamsMod  , only: CNParamsShareInst, anoxia_wtsat, nlev_soildecomp_standard
...
mino2lim = CNParamsShareInst%mino2lim
```

3. The routines for both shared and private variables are called from readParamsMod.F90

Reading parameters - readParamsMod.F90

```
module readParamsMod
...
  subroutine CNParamsReadFile ()
    !
    ! read CN and BGC shared parameters
    !
    use CNAllocationMod         , only : readCNAllocParams
    use CNDecompMod             , only : readCNDecompParams
    use CNDecompCascadeBGCMod   , only : readCNDecompBgcParams
    use CNDecompCascadeCNMod    , only : readCNDecompCnParams
    use CNPhenologyMod          , only : readCNPhenolParams
    use CNMRespMod              , only : readCNMRespParams
    use CNNDynamicsMod          , only : readCNNDynamicsParams
    use CNGapMortalityMod       , only : readCNGapMortParams
    use CNNitrifDenitrifMod     , only : readCNNitrifDenitrifParams
    use CNSoilLittVertTranspMod , only : readCNSoilLittVertTranspParams
    use CNSharedParamsMod       , only : CNParamsReadShared,CNParamsShareInst
    use ch4Mod                  , only : readCH4Params
...
    ! read the file with parameters
    if (masterproc) then
       write(iulog,*) 'readParamsMod.F90::'//trim(subname)//' :: reading CN '//&
          'and BGC parameter file'
    end if
...
! read shared parameters
 call CNParamsReadShared(ncid)
! read private parameters based on use case
    if (use_cn) then
       !
       ! populate each module with private parameters
       !
       call readCNAllocParams(ncid)
       call readCNDecompParams(ncid)
       if (use_century_decomp) then
          call readCNDecompBgcParams(ncid)
       else
          call readCNDecompCnParams(ncid)
       end if
       call readCNPhenolParams(ncid)
       call readCNMRespParams (ncid)
       call readCNNDynamicsParams (ncid)
       call readCNGapMortParams (ncid)
       if (use_nitrif_denitrif) then
          call readCNNitrifDenitrifParams(ncid)
       end if
...
       call readCNSoilLittVertTranspParams(ncid)
...
       if (use_lch4) then
          call readCH4Params (ncid)
       end if
...
    end if
...
  end subroutine CNParamsReadFile
end module readParamsMod
```

Associate refactor

- Please do not use pointers as arguments to a subroutine if that pointer is not placed in a associate block. Using an Associate Block removes one declaration for each pointer, makes the code easier to modify, makes the code more robust and sets us up for future interface refactorings. Refer also to the section above .

Associate Refactor

```
!before the Associate Refactor, pointers were delcared and assigned as follows:
-    real(r8), pointer :: fdry(:)           ! fraction of foliage that is green and dry [-] (new)
-    fdry => pps%fdry
! after the refactor, the pointer declaration is removed and handled in the associate statement.
+   associate(&
+   fdry               => pps%fdry             & ! Output: [real(r8) (:)]  fraction of foliage that is
green and dry [-] (new)
+   )
 ...
+    end associate
+   end subroutine FracWet
```

Refactor clmtype

- We have flattened the structure of clmtype so that most variables only require one dereference. You will immediately notice if you have used the old style as your code will not compile. An example is shown below.

Refactor clmtype

```
call hist_addfld1d (fname='SNOOCFRCL', units='W/m^2', &
          avgflag='A', long_name='surface forcing of OC in snow (land) ', &
-         ptr_pft=clm3%g%l%c%p%pef%sfc_frc_oc, set_urb=spval)
+         ptr_pft=pef%sfc_frc_oc, set_urb=spval)
```

Preprocessor macro removal

- We are in the process of removing all preprocessor macros (CPP tokens) from the CLM code. Doing so will allow us to use one binary in all of our test suite and greatly speed up the development and testing process. There are two steps in the process:
    1. Remove most ifdef declarations from the code and replace with logical variables in main/controlMod.F90.

Macro removal - Step 1

```
#if (defined LCH4)
     use_lch4 = .true.
#endif
#if (defined NITRIF_DENITRIF)
     use_nitrif_denitrif = .true.
#endif
#if (defined VERTSOILC)
     use_vertsoilc = .true.
#endif
#if (defined EXTRALAKELAYERS)
     use_extralakelayers = .true.
#endif
#if (defined VICHYDRO)
     use_vichydro = .true.
#endif
#if (defined CENTURY_DECOMP)
     use_century_decomp = .true.
#endif
#if (defined CN)
     use_cn = .true.
#endif
#if (defined CNDV)
     use_cndv = .true.
#endif
#if (defined CROP)
     use_crop = .true.
#endif
#if (defined SNICAR_FRC)
     use_snicar_frc = .true.
#endif
#if (defined VANCOUVER)
     use_vancouver = .true.
#endif
#if (defined MEXICOCITY)
     use_mexicocity = .true.
#endif
#if (defined NOIO)
     use_noio = .true.
#endif
```

2. The final portion of this work (which will be in an upcoming CLM tag), removes all of the remaining ifdefs and replaces this functionality via namelist variables.

Macro removal - Step 2

```
Fill in when Ben makes this tag
```

The bounds type

- The introduction of the bounds type cleans up CLM interfaces since we don't have to pass begc, endc, etc... explicitly. Currently, only the bounds type is passed and then members are de-referenced in a routine when you need to use them. If you don't want to use this

```
bounds%begc
```

you can place the following in an associate statement.

```
begc => bounds%begc
```

Doing so may save you some refactoring work in the body of the subroutine or function you are working with.

CLM without the bounds type

```
use decompMod, only : get_proc_bounds
...
integer :: begp, endp   ! per-proc beginning and ending pft indices
integer :: begc, endc   ! per-proc beginning and ending column indices
integer :: begl, endl   ! per-proc beginning and ending landunit indices
integer :: begg, endg   ! per-proc gridcell ending gridcell indices
...
call get_proc_bounds(begg, endg, begl, endl, begc, endc, begp, endp)
do c = begc, endc
```

CLM with the bounds type

```
type(bounds_type), intent(in) :: bounds  ! bounds
...
do c = bounds%begc, bounds%endc
```