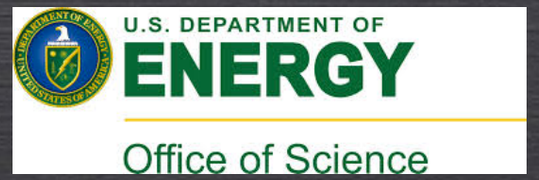
Recent CLM Refactoring

Bill Sacks (NCAR)

Much of the work described here was done by Mariana Vertenstein

with contributions from:
Ben Andre, Erik Kluzek, Charlie Koven, Dave Lawrence,
Stefan Muszala, Sean Santos, Jinyun Tang, and others

Appendix slides contain additional reference material



Refactoring for Greater Modularity & Object Orientation

Old Centralized Modules – Now Gone

- Data types
  - clmtype.F90
  - clmtypeInitMod.F90
- Initialization
  - initTimeConst.F90
  - initCold.F90
- History
  - histFldsMod.F90
- Accumulation
  - accumulMod.F90
- Restart
  - biogeophysRestMod.F90
  - CNRestMod.F90
- Biogeochemistry
  - CNSetValue

Example of New Modularization: IrrigationMod.F90

```
module IrrigationMod

  ! ...

  type, public :: irrigation_type
     private
     ! Public data members
     ! Note: these should be treated as read-only by other modules
     real(r8), pointer, public :: qflx_irrig_patch(:) ! patch irrigation flux (mm H2O/s)
     real(r8), pointer, public :: qflx_irrig_col  (:) ! col irrigation flux (mm H2O/s)

     ! Private data members; set in initialization:
     type(irrigation_params_type) :: params
     integer :: dtime                     ! land model time step (sec)
     integer :: irrig_nsteps_per_day ! number of time steps per day in which we irrigate
     real(r8), pointer :: relsat_so_patch(:,:) ! ... [patch, nlevgrnd]

     ! Private data members; time-varying:
     real(r8), pointer :: irrig_rate_patch        (:)  ! current irrigation rate [mm/s]
     integer , pointer :: n_irrig_steps_left_patch (:)  ! number of time steps ...

  contains
    ! type definition continued on next slide...
```

Example of New Modularization: IrrigationMod.F90

```
type, public :: irrigation_type
   ! ...

 contains
   ! Public routines
   procedure, public :: Init
   procedure, public :: Restart
   procedure, public :: ApplyIrrigation
   procedure, public :: CalcIrrigationNeeded

   ! Public simply to support unit testing; should not be used from CLM code
   procedure, public, nopass :: IrrigationDeficit ! compute the irrigation deficit ...

   ! Private routines
   procedure, private :: InitAllocate
   procedure, private :: InitHistory
   procedure, private :: InitCold
   procedure, private :: CalcIrrigNstepsPerDay   ! ...
   procedure, private :: PointNeedsCheckForIrrig ! ...
end type irrigation_type
```

Example of New Modularization: IrrigationMod.F90

```
!------------------------------------------------------------------------
subroutine Init(this, bounds, soilstate_inst, soil_water_retention_curve)
  class(irrigation_type) , intent(inout) :: this
  type(bounds_type)      , intent(in)    :: bounds
  type(soilstate_type)   , intent(in)    :: soilstate_inst
  class(soil_water_retention_curve_type), intent(in) :: soil_water_retention_curve

  call this%InitAllocate(bounds)
  call this%InitHistory(bounds)
  call this%InitCold(bounds, soilstate_inst, soil_water_retention_curve)
end subroutine Init

!------------------------------------------------------------------------
subroutine InitAllocate(this, bounds)
  ! ...

  allocate(this%qflx_irrig_patch(begp:endp))          ; this%qflx_irrig_patch(:)   = nan
  allocate(this%qflx_irrig_col  (begc:endc))          ; this%qflx_irrig_col  (:)   = nan
  allocate(this%relsat_so_patch (begp:endp,nlevgrnd)) ; this%relsat_so_patch (:,:) = nan
  allocate(this%irrig_rate_patch(begp:endp))          ; this%irrig_rate_patch(:)   = nan
end subroutine InitAllocate
```

Example of New Modularization: IrrigationMod.F90

```
!-----------------------------------------------------------------------
subroutine CalcIrrigationNeeded(this, bounds, num_exposedvegp, filter_exposedvegp, &
     time_prev, elai, btran, rootfr, t_soisno, eff_porosity, h2osoi_liq)
  ! ...

  do f = 1, num_exposedvegp
     p = filter_exposedvegp(f)
     g = patch%gridcell(p)
     check_for_irrig(p) = this%PointNeedsCheckForIrrig( &
          pft_type=patch%itype(p), elai=elai(p), btran=btran(p), &
          time_prev=time_prev, londeg=grc%londeg(g))

     if (check_for_irrig(p)) then
        this%n_irrig_steps_left_patch(p) = this%irrig_nsteps_per_day
        this%irrig_rate_patch(p)         = 0._r8  ! reset; we'll add to this later
     end if
  end do

  ! ...
end subroutine CalcIrrigationNeeded
```

More Common for Existing Code: Semi-Modularity

- New *Type.F90 modules combine variable declarations with infrastructure code – what used to be in:
  - clmtype.F90
  - clmtypeInitMod.F90
  - initCold.F90
  - histFldsMod.F90
  - *RestMod.F90
  - (and others)
- But science routines are in separate modules
- Example: TemperatureType.F90 (see appendix)

Why is this Good?

- Explicit arguments show data flow through the system
- Easier to read & modify code: No longer need to touch numerous infrastructure modules
- Supports unit testing
- Supports having multiple implementations of a parameterization

src/main/clm_instMod.F90

```
module clm_instMod
  ! ...
  !-----------------------------------------
  ! Instances of component types
  !-----------------------------------------
  ! ...
  type(irrigation_type) :: irrigation_inst
  ! ...
contains
  subroutine clm_instInit(bounds)
    ! ...
    call irrigation_inst%init(bounds, soilstate_inst, soil_water_retention_curve)
    ! ...
  end subroutine clm_instInit

  subroutine clm_instRest(bounds, ncid, flag)
    ! ...
    call irrigation_inst%restart (bounds, ncid, flag=flag)
    ! ...
  end subroutine clm_instRest

end module clm_instMod
```

Instances only used directly by clm_initializeMod and clm_driver

Separation of below-ground and above-ground biogeochemistry

- Goals:
  - Make soil biogeochemistry independent of CN or ED vegetation biogeochemistry
  - Separate ED and CN functionality – EITHER ED or CN is on and both will work with the same soil biogeochemistry
- Directory structure:
  - soilbiogeochem/ – new; independent of ED or CN
  - biogeochem/ – CN vegetation
  - ED/ – ED vegetation

Supporting Alternative Implementations via Polymorphism

Martin Fowler (*Refactoring: Improving the Design of Existing Code*, pp 255-256): "One of the grandest sounding words in object jargon is *polymorphism*.... it allows you to avoid writing an explicit conditional when you have objects whose behavior varies depending on their types [in CLM: when you have science implementations whose behavior varies depending on a namelist flag]. The biggest gain occurs when this same set of conditions appears in many places in the program. If you want to add a new [implementation], you have to find and update all the conditionals. But with [polymorphism] you just create a new subclass and provide the appropriate methods.... [This] reduces the dependencies in your system and makes it easier to update."

Supporting Alternative Implementations via Polymorphism

- A base type defines the common interface
  - Routines called from driver or elsewhere
  - Variables available to other parts of the code
- Separate module for each implementation
  - Implementation of each routine
  - Private data specific to this implementation
- Examples:
  - Ozone on vs. off: See appendix
  - Soil water retention curve
  - Nutrient competition method

Other Useful Stuff

Development with Unit Tests

- Leverages new unit testing framework in CESM
  - Uses pFUnit
  - CESM infrastructure developed by Sean Santos

```
@Test
subroutine no_irrigation_for_frozen_soil(this)
  class(TestIrrigation), intent(inout) :: this

  ! Setup
  call setupIrrigation(this%irrigation_inputs, this%irrigation, maxpft=1)
  this%irrigation_inputs%t_soisno(bounds%begc, :) = 272._r8

  ! Call irrigation routines
  call this%irrigation_inputs%calculateAndApplyIrrigation(this%irrigation, this%numf, this%filter)

  ! Check result
  @assertEqual(0._r8, this%irrigation%qflx_irrig_patch(bounds%begp))

end subroutine no_irrigation_for_frozen_soil
```

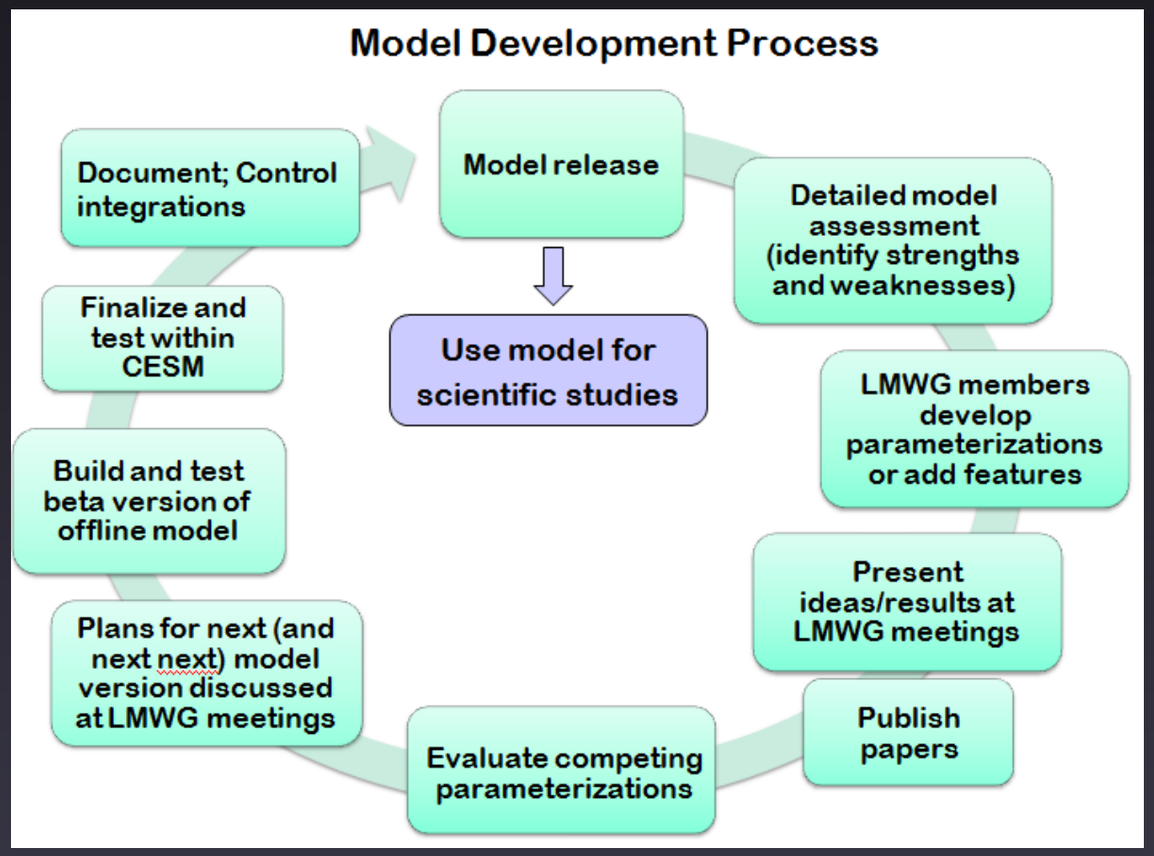
- Contact us if you'd like help developing unit tests for your code
  - clm-cmt@cgd.ucar.edu

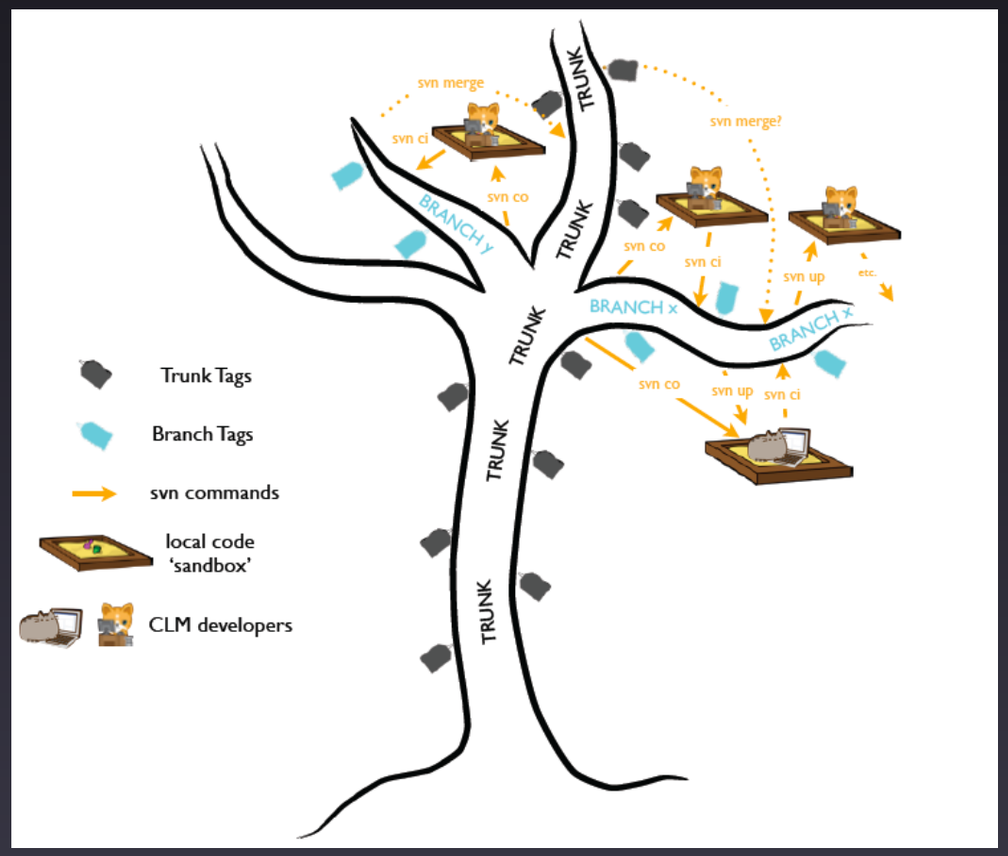
CLM Developers' Guide

http://www2.cesm.ucar.edu/working-groups/lmwg/developer-guidelines

Software development guidelines

- Software developer's guide: read this for general information on the steps in the model development process including information on coding standards, maintaining a branch, testing, and working with the CLM Code Management Team
  - Coding practices
  - Using SVN to work with development branches
  - CLM testing
  - Upcoming CLM branch and trunk tags
  - Recent CLM code refactoring





Please Contact Us!

- The CLM Code Management Team (CLM-CMT) is here to help
  - clm-cmt@cgd.ucar.edu
- We welcome any feedback
- We encourage you to contact us *before* starting big developments

Appendix –
Additional Reference Material

More Details on the Refactor for Modularity

- Data structures arranged by scientific functional categories
  - temperature_type, waterstate_type, energyflux_type, ….
- All subgrid levels are in the data structure
  - variables are now appended with a unique suffix to indicate their subgrid levels
  - new suffixes: _patch, _col, _lun, _grc
  - This does NOT effect the science code base – ONLY the associate statements
- Separate module for each data type definition
  - TemperatureType.F90, WaterstateType.F90, EnergyFluxType.F90,…
- Each data type has associated methods for
  - Allocation of variables – *all* variables now initialized as NaNs
  - Cold start initialization of variables – this is now ALWAYS done and overwritten if finidat is read in as spun up dataset (also now have online interpolation of initial conditions as part of this refactor as well)
  - History initialization of variables (all history fields now initialized as spval)
  - Restart initialization of variables
  - Accumulation initialization and accumulation update of variables
- Instantiation of datatypes is now separate from their declaration (now in clm_instMod.F90)

Recent File Renaming

- Biogeophysics1Mod.F90 => CanopyTemperatureMod.F90
- Hydrology1Mod => CanopyHydrologyMod
- Biogeophysics2Mod => SoilFluxesMod
- HydrologyNoDrainage, HydrologyDrainage => SoilHydrologyMod
- PhotosynthesisMod has been separated from CanopyFluxesMod – and is its own module

Example of New, More Common Semi-Modularization: TemperatureType.F90

This module contains all infrastructure code that operates on temperature variables, but does NOT contain science routines.

```
module TemperatureType
  ! ...
  type, public :: temperature_type
     real(r8), pointer :: t_veg_patch (:)   ! patch vegetation temperature (Kelvin)
     real(r8), pointer :: t_soisno_col(:,:) ! col soil temperature (Kelvin)  (-nlevsno+1:nlevgrnd)
     real(r8), pointer :: t_grnd_col  (:)   ! col ground temperature (Kelvin)
     real(r8), pointer :: taf_lun     (:)   ! lun urban canopy air temperature (K)
     ! ...
   contains

     procedure, public  :: Init
     procedure, public  :: Restart
     procedure, private :: InitAllocate
     procedure, private :: InitHistory
     procedure, private :: InitCold
     procedure, public  :: InitAccBuffer
     procedure, public  :: InitAccVars
     procedure, public  :: UpdateAccVars

end type temperature_type
!------------------------------------------------------------------------
```

Example of New, More Common Semi-Modularization: TemperatureType.F90

```
!------------------------------------------------------------------------
subroutine Init(this, bounds, &
     em_roof_lun,  em_wall_lun, em_improad_lun, em_perroad_lun, &
     is_simple_buildtemp, is_prog_buildtemp)
  !
  ! !DESCRIPTION:
  !
  ! Initialization of the data type. Allocate data, setup variables
  ! for history output, and initialize values needed for a cold-start.
  !
  class(temperature_type)         :: this
  type(bounds_type) , intent(in) :: bounds
  real(r8)          , intent(in) :: em_roof_lun(bounds%begl:)
  real(r8)          , intent(in) :: em_wall_lun(bounds%begl:)
  real(r8)          , intent(in) :: em_improad_lun(bounds%begl:)
  real(r8)          , intent(in) :: em_perroad_lun(bounds%begl:)
  logical           , intent(in) :: is_simple_buildtemp  ! Simple building temp is being used
  logical           , intent(in) :: is_prog_buildtemp    ! Prognostic building temp is being used

  call this%InitAllocate ( bounds )
  call this%InitHistory ( bounds, is_simple_buildtemp, is_prog_buildtemp )
  call this%InitCold ( bounds,                    &
       em_roof_lun(bounds%begl:bounds%endl),    &
       em_wall_lun(bounds%begl:bounds%endl),    &
       em_improad_lun(bounds%begl:bounds%endl), &
       em_perroad_lun(bounds%begl:bounds%endl), &
       is_simple_buildtemp, is_prog_buildtemp)

end subroutine Init
! ...
```

Polymorphism Example: Ozone

- There are two options for ozone: on & off
  - Ozone off can be thought of as an alternative (albeit very simple) implementation
- Without polymorphism, there were a number of conditionals ("if (use_ozone) then ...") throughout the code, both inside and outside the ozone module. This made it more difficult to understand what code applies and what doesn't apply when ozone is off.
- The polymorphism implementation allows the use_ozone conditional to appear in only one place in the code.
- There are then separate modules that provide the implementation for ozone on and ozone off.

Polymorphism Example: Ozone

Base class provides the common interface, as well as routines that are shared between all implementations (e.g., allocating and initializing public data).

```
module OzoneBaseMod

  !-----------------------------------------------------------------------
  ! !DESCRIPTION:
  ! Define the interface for ozone_type, which calculates ozone-induced stress. The type
  ! defined here is abstract; it will get instantiated as a concrete type that extends
  ! this base type (e.g., an ozone-off or ozone-on version).
  ! ...
  ! !PUBLIC TYPES:
  type, abstract, public :: ozone_base_type
     private

     ! Public data members
     ! These should be treated as read-only by other modules (except that they can be
     ! modified by extensions of the ozone_base_type)
     real(r8), pointer, public :: o3coefvsha_patch(:)  ! ozone coefficient for photosynthesis, shaded leaves (0 - 1)
     real(r8), pointer, public :: o3coefvsun_patch(:)  ! ozone coefficient for photosynthesis, sunlit leaves (0 - 1)
     real(r8), pointer, public :: o3coefgsha_patch(:)  ! ozone coefficient for conductance, shaded leaves (0 - 1)
     real(r8), pointer, public :: o3coefgsun_patch(:)  ! ozone coefficient for conductance, sunlit leaves (0 - 1)

   contains
     ! The following routines need to be implemented by all type extensions
     procedure(Init_interface)            , public, deferred :: Init
     procedure(Restart_interface)         , public, deferred :: Restart
     procedure(CalcOzoneStress_interface) , public, deferred :: CalcOzoneStress

     ! The following routines should only be called by extensions of the ozone_base_type
     procedure, public :: InitAllocateBase
     procedure, public :: InitColdBase

  end type ozone_base_type
```

Polymorphism Example: Ozone

Base class, continued

```
abstract interface

   subroutine Init_interface(this, bounds)
     use decompMod, only : bounds_type
     import :: ozone_base_type

     class(ozone_base_type), intent(inout) :: this
     type(bounds_type), intent(in) :: bounds
   end subroutine Init_interface

   subroutine Restart_interface(this, bounds, ncid, flag)
     use decompMod , only : bounds_type
     use ncdio_pio , only : file_desc_t
     import :: ozone_base_type

     class(ozone_base_type)             :: this
     type(bounds_type) , intent(in)     :: bounds
     type(file_desc_t) , intent(inout) :: ncid ! netcdf id
     character(len=*)  , intent(in)     :: flag ! 'read', 'write' or 'define'
   end subroutine Restart_interface

   subroutine CalcOzoneStress_interface(this, bounds, num_exposedvegp, filter_exposedvegp, &
        forc_pbot, forc_th, rssun, rssha, rb, ram, tlai)
     use decompMod    , only : bounds_type
     use shr_kind_mod , only : r8 => shr_kind_r8
     import :: ozone_base_type

     class(ozone_base_type) , intent(inout) :: this
     type(bounds_type)      , intent(in)    :: bounds
     integer  , intent(in) :: num_exposedvegp           ! number of points in filter_exposedvegp
     integer  , intent(in) :: filter_exposedvegp(:)     ! patch filter for non-snow-covered veg
     real(r8) , intent(in) :: forc_pbot( bounds%begc: ) ! atmospheric pressure (Pa)
     real(r8) , intent(in) :: forc_th( bounds%begc: )   ! atmospheric potential temperature (K)
     real(r8) , intent(in) :: rssun( bounds%begp: )     ! leaf stomatal resistance, sunlit leaves (s/m)
     real(r8) , intent(in) :: rssha( bounds%begp: )     ! leaf stomatal resistance, shaded leaves (s/m)
     real(r8) , intent(in) :: rb( bounds%begp: )        ! boundary layer resistance (s/m)
     real(r8) , intent(in) :: ram( bounds%begp: )       ! aerodynamical resistance (s/m)
     real(r8) , intent(in) :: tlai( bounds%begp: )      ! one-sided leaf area index, no burying by snow
   end subroutine CalcOzoneStress_interface

end interface
```

Polymorphism Example: Ozone

OzoneMod provides the implementation when ozone is turned on

```
module OzoneMod
  ! ...
  type, extends(ozone_base_type), public :: ozone_type
     private
     ! Private data members
     real(r8), pointer :: o3uptakesha_patch(:) ! ozone dose, shaded leaves (mmol O3/m^2)
     real(r8), pointer :: o3uptakesun_patch(:) ! ozone dose, sunlit leaves (mmol O3/m^2)

     real(r8), pointer :: tlai_old_patch(:)  ! tlai from last time step
   contains
     ! Public routines
     procedure, public :: Init
     procedure, public :: Restart
     procedure, public :: CalcOzoneStress

     ! Private routines
     procedure, private :: InitAllocate
     procedure, private :: InitHistory
     procedure, private :: InitCold

     ! Calculate ozone stress for a single point, for just sunlit or shaded leaves
     procedure, private, nopass :: CalcOzoneStressOnePoint
end type ozone_type

! Implementation follows. This can be implemented assuming that ozone is turned on.
```

Polymorphism Example: Ozone

OzoneOffMod provides the implementation when ozone is turned off

```
module OzoneOffMod

  !-----------------------------------------------------------------------
  ! !DESCRIPTION:
  ! Provides an implementation of ozone_base_type for the ozone-off case. Note that very
  ! little needs to be done in this case, so this module mainly provides empty
  ! implementations to satisfy the interface.
  ! ...
  type, extends(ozone_base_type), public :: ozone_off_type
     private
   contains
     procedure, public :: Init
     procedure, public :: Restart
     procedure, public :: CalcOzoneStress
  end type ozone_off_type

contains

  subroutine Init(this, bounds)
    class(ozone_off_type) , intent(inout) :: this
    type(bounds_type)     , intent(in)    :: bounds

    call this%InitAllocateBase(bounds)
    call this%InitColdBase(bounds)
  end subroutine Init

  subroutine Restart(this, bounds, ncid, flag)
    use ncdio_pio , only : file_desc_t

    class(ozone_off_type) :: this
    type(bounds_type), intent(in) :: bounds
    type(file_desc_t) , intent(inout) :: ncid ! netcdf id
    character(len=*)  , intent(in)    :: flag ! 'read', 'write' or 'define'

    ! DO NOTHING

  end subroutine Restart
```

Polymorphism Example: Ozone

OzoneOffMod, continued

```
subroutine CalcOzoneStress(this, bounds, num_exposedvegp, filter_exposedvegp, &
       forc_pbot, forc_th, rssun, rssha, rb, ram, tlai)

  class(ozone_off_type) , intent(inout) :: this
  type(bounds_type)     , intent(in)    :: bounds
  integer  , intent(in) :: num_exposedvegp          ! number of points in filter_exposedvegp
  integer  , intent(in) :: filter_exposedvegp(:)    ! patch filter for non-snow-covered veg
  real(r8) , intent(in) :: forc_pbot( bounds%begc: ) ! atmospheric pressure (Pa)
  real(r8) , intent(in) :: forc_th( bounds%begc: )   ! atmospheric potential temperature (K)
  real(r8) , intent(in) :: rssun( bounds%begp: )     ! leaf stomatal resistance, sunlit leaves (s/m)
  real(r8) , intent(in) :: rssha( bounds%begp: )     ! leaf stomatal resistance, shaded leaves (s/m)
  real(r8) , intent(in) :: rb( bounds%begp: )        ! boundary layer resistance (s/m)
  real(r8) , intent(in) :: ram( bounds%begp: )       ! aerodynamical resistance (s/m)
  real(r8) , intent(in) :: tlai( bounds%begp: )      ! one-sided leaf area index, no burying by snow

  ! Explicitly set outputs to 1. This isn't really needed, because they should still be
  ! at 1 from cold-start initialization, but do this for clarity here.

  this%o3coefvsha_patch(bounds%begp:bounds%endp) = 1._r8
  this%o3coefvsun_patch(bounds%begp:bounds%endp) = 1._r8
  this%o3coefgsha_patch(bounds%begp:bounds%endp) = 1._r8
  this%o3coefgsun_patch(bounds%begp:bounds%endp) = 1._r8

end subroutine CalcOzoneStress
```

Polymorphism Example: Ozone

OzoneFactoryMod creates the appropriate instance of ozone_base_type. This is the only place in the code where there is a conditional based on use_ozone.

```
module OzoneFactoryMod

  !-----------------------------------------------------------------------
  ! !DESCRIPTION:
  ! Factory to create an instance of ozone_base_type. This module figures out the
  ! particular type to return.
  ! ...
contains

  !-----------------------------------------------------------------------
  function create_and_init_ozone_type(bounds) result(ozone)
    !
    ! !DESCRIPTION:
    ! Create and initialize an object of ozone_base_type, and return this object. The
    ! particular type is determined based on the use_ozone namelist parameter.
    !
    ! !USES:
    use clm_varctl    , only : use_ozone
    use OzoneBaseMod , only : ozone_base_type
    use OzoneOffMod  , only : ozone_off_type
    use OzoneMod      , only : ozone_type
    !
    ! !ARGUMENTS:
    class(ozone_base_type), allocatable :: ozone  ! function result
    type(bounds_type), intent(in) :: bounds
    !-----------------------------------------------------------------------

    if (use_ozone) then
       allocate(ozone, source = ozone_type())
    else
       allocate(ozone, source = ozone_off_type())
    end if

    call ozone%Init(bounds)

  end function create_and_init_ozone_type

end module OzoneFactoryMod
```

Polymorphism Example: Ozone

Other modules can refer to subroutines and variables in ozone_base_type, without any concern for whether ozone is on or off in this run (thus decoupling and simplifying different parts of the code).

```
module CanopyFluxesMod
  ! ...
  use OzoneBaseMod          , only : ozone_base_type
  ! ...
  !------------------------------------------------------------------------
  subroutine CanopyFluxes(bounds,  num_exposedvegp, filter_exposedvegp, &
       ed_allsites_inst,  atm2lnd_inst, canopystate_inst, cnveg_state_inst,              &
       energyflux_inst, frictionvel_inst, soilstate_inst, solarabs_inst, surfalb_inst, &
       temperature_inst, waterflux_inst, waterstate_inst, ch4_inst, ozone_inst, photosyns_inst, &
       humanindex_inst, soil_water_retention_curve)
  ! ...
  class(ozone_base_type)                    , intent(inout)         :: ozone_inst
  ! ...
  call ozone_inst%CalcOzoneStress( &
       bounds, fn, filterp, &
       forc_pbot = atm2lnd_inst%forc_pbot_downscaled_col(bounds%begc:bounds%endc), &
       forc_th   = atm2lnd_inst%forc_th_downscaled_col(bounds%begc:bounds%endc), &
       rssun     = photosyns_inst%rssun_patch(bounds%begp:bounds%endp), &
       rssha     = photosyns_inst%rssha_patch(bounds%begp:bounds%endp), &
       rb        = frictionvel_inst%rb1_patch(bounds%begp:bounds%endp), &
       ram       = frictionvel_inst%ram1_patch(bounds%begp:bounds%endp), &
       tlai      = canopystate_inst%tlai_patch(bounds%begp:bounds%endp))
```

Polymorphism Example: Ozone

Modules referring to ozone, continued

```
module  PhotosynthesisMod
  ! ...
  use OzoneBaseMod          , only : ozone_base_type
  ! ...
  !-----------------------------------------------------------------------
  subroutine Photosynthesis ( bounds, fn, filterp, &
       esat_tv, eair, oair, cair, rb, btran, &
       dayl_factor, atm2lnd_inst, temperature_inst, surfalb_inst, solarabs_inst, &
       canopystate_inst, ozone_inst, photosyns_inst, phase)
! ...
o3coefv    =>    ozone_inst%o3coefvsun_patch ! ...
o3coefg    =>    ozone_inst%o3coefgsun_patch ! ...
! ...
rs_z(p,iv) = min(1._r8/gs, rsmax0)
rs_z(p,iv) = rs_z(p,iv) / o3coefg(p)

psn_z(p,iv) = ag(p,iv)
psn_z(p,iv) = psn_z(p,iv) * o3coefv(p)
```