Community Climate System Model
National Center for Atmospheric Research, Boulder, CO

CLM2.0 Code Reference

Version 1.0 *Mariana Vertenstein, Sam Levis, Keith Oleson, and Peter Thornton*

Contents

1 Code Structure

1.1 Calling Tree

The following is a brief outline of the calling sequence for the main CLM2.0 driver routine, driver.F90. A comprehensive outline of the calling sequence for the full CLM2.0 model is provided in the accompanying html document, CLM2.0 Comprehensive Calling Sequence.

- -> histend
- -> get_curr_date
- -> interpMonthlyVeg
  - -> readMonthlyVegetation
- -> *****begin first loop over patch points****
- -> Hydrology1
  - -> Fwet
- -> Biogeophysics1
  - -> QSat
  - -> SurfaceRadiation
  - -> BareGroundFluxes
    - -> MoninObukIni
    - -> FrictionVelocity
  - -> CanopyFluxes
    - -> Qsat
    - -> MoninObukIni
    - -> FrictionVelocity
    - -> Stomata (call for sunlit leaves and shaded leaves)
    - -> SensibleHCond
    - -> LatentHCond
    - -> Qsat
- -> Biogeophysics_Lake
  - -> SurfaceRadiation
  - -> Qsat
  - -> MoninObukIni
  - -> FrictionVelocity
  - -> Qsat
  - -> Tridiagonal
- -> EcosystemDyn
- -> SurfaceAlbedo
  - -> shr_orb_decl

  - -> shr_orb_cosz
  - -> SnowAlbedo (called for direct beam and diffuse beam)
  - -> SoilAlbedo
  - -> TwoStream (called for visible direct, visible diffuse, NIR direct and NIR diffuse)

- -> Biogeophysics2
  - -> SoilTemperature
    - -> SoilThermalProp
    - -> Tridiagonal
    - -> PhaseChange
- -> *****end first loop over patch points*****
- -> *****begin second loop over patch points*****
- -> Hydrology2
  - -> SnowWater
  - -> SurfaceRunoff
  - -> Infiltration
  - -> SoilWater
    - -> Tridiagonal
  - -> Drainage
  - -> SnowCompaction
  - -> CombineSnowLayers
    - -> Combo
  - -> DivideSnowLayers
    - -> Combo
  - -> WetIceHydrology
- -> Hydrology_Lake
- -> SnowAge
- -> BalanceCheck
- -> *****end second loop over patch points*****
- -> histUpdate
- -> Rtmriverflux
  - -> Rtm
- -> histhandler
- -> restwrt
- -> inicwrt

1.2 Code Flow - Main Interface

The CLM2.0 model can be built to run in one of three modes. It can run as a stand alone executable where atmospheric forcing data is periodically read . This will be referred to as offline mode. It can also be run as part of the Community Atmosphere Model (CAM) where communication between the atmospheric and land models occurs via subroutine calls. This will be referred to as cam mode. Finally, it can be run as a component in a system of geophysical models (CCSM). In this mode, the atmosphere, land (CLM2), ocean and sea-ice models are run as separate executables that communicate with each other via the CCSM flux coupler. This will be referred to as csm mode.

offline mode: The routine, **program_off.F90**, provides the program interface for running CLM2.0 in offline mode. This routine first initializes the CLM2.0 model. Part of this initialization consists of the the determination of orbital parameters by a call to the routine **shr_orb_params.F90**. Subsequently, the time stepping loop of the model is executed by obtaining atmospheric forcing and calling the CLM2.0 driver.

csm mode: The routine, **program_csm.F90**, provides the program interface for running CLM2.0 in csm mode. In this mode, orbital parameters are obtained from the flux coupler during initialization whereas atmospheric data are obtained from the flux coupler during the time stepping loop.

cam mode: The module, **atm_lndMod.F90**, contains the subroutine interfaces necessary to run CLM2.0 in cam mode. The model is initialized by a call to subroutine **atmlnd_ini** whereas subroutine **atmlnd_drv** is called at every time step by the CAM atmospheric model to update the land state and return the necessary states and fluxes back to the atmosphere.

1.3 Code Flow - Driver Loop

The following presents a brief outline of the routines appearing in the calling sequence for **program_off.F90**, the main CLM2.0 offline program. Most of the following routines are invoked from the driver routine, **driver.F90**.

shr_orb_params: Calculate Earth's orbital parameters.

Initialize: Calls a series of subroutines (see CLM2.0 Comprehensive Calling Sequence) which initialize model parameters, read and/or create a surface dataset, read the initial file (initial simulations only), read the restart file (restart or branch simulations only). If no initial dataset is specified in the namelist, the model uses an internal initialization. If no surface dataset is specified in the namelist, the model uses a list of raw datasets to create a surface dataset and read it in.

atmdrv: Read in atmospheric fields and generate atmospheric forcing.

driver: Driver for CLM2.0 physics.

histend: Determine if current time step is the end of history interval.

get_curr_date: Determine calender information for next time step.

interpMonthlyVeg: Determine if two new months of vegetation data need to be read in.

readMonthlyVegetation: Read monthly vegetation data for two consecutive months.

Hydrology1: Calculation of (1) water storage of intercepted precipitation (2) direct throughfall and canopy drainage of precipitation (3) the fraction of foliage covered by water and the fraction of foliage that is dry and transpiring and (4) snow layer initialization if the snow accumulation exceeds 10 mm.

Fwet: Determine the fraction of foliage covered by water and the fraction of foliage that is dry and transpiring.

Biogeophysics1: Main subroutine to determine leaf temperature and surface fluxes based on ground temperature from previous time step.

QSat: Compute saturation mixing ratio and the change in saturation mixing ratio with respect to temperature.

SurfaceRadiation: Compute Visible and NIR solar fluxes absorbed by vegetation and ground surface. Split canopy absorption into sunlit and shaded canopy. Calculate NDVI and reflected solar radiation. This routine is also used for surface radiation for lake biogeophysics.

BareGroundFluxes: Compute sensible and latent heat fluxes and their derivatives with respect to ground temperature using ground temperatures from previous time step for non-vegetated surfaces or snow-covered vegetation. Calculate stability and aerodynamic resistances.

MoninObukIni: Initialize Monin-Obukhov length.

FrictionVelocity: Calculation of the friction velocity and the relation for potential temperature and humidity profiles of surface boundary layer.

CanopyFluxes: Calculates the leaf temperature, leaf fluxes, transpiration, photosynthesis and updates the dew accumulation due to evaporation.

Stomata: Leaf stomatal resistance and leaf photosynthesis. Uses Ball-Berry formulation for stomatal conductance, and Farquhar photosynthesis model.

SensibleHCond: Provides dimensional and non-dimensional sensible heat conductances for canopy and soil flux calculations.

LatentHCond: Provides dimensional and non-dimensional latent heat conductances for canopy and soil flux calculations.

Biogeophysics_Lake: Calculates lake temperatures and surface fluxes. Lake temperatures are determined from a one-dimensional thermal stratification model based on eddy diffusion concepts to represent vertical mixing of heat.

EcosystemDyn: Determine vegetation phenology

SurfaceAlbedo: Surface albedos, fluxes (per unit incoming direct and diffuse radiation) reflected, transmitted, and absorbed by vegetation, and sunlit fraction of the canopy.

shr_orb_decl: Determine solar declination for next time step.

shr_orb_cosz: Determine cosine of solar zenith angle for next time step.

SnowAlbedo: Determine direct and diffuse visible and NIR snow albedos.

SoilAlbedo: Determine soil/lake/glacier/wetland albedos.

TwoStream: Use two-stream approximation to calculate visible and NIR fluxes absorbed by vegetation, reflected by vegetation, and transmitted through vegetation for unit incoming direct or diffuse flux given an underlying surface with known albedo.

Biogeophysics2: Main subroutine to determine soil/snow temperatures including ground surface temperature and update surface fluxes for new ground temperature.

SoilTemperature: Determine soil/snow temperatures including ground surface temperature.

SoilThermProp: Determine soil/snow thermal conductivity and heat capacity.

Tridiagonal: Solve tridiagonal system of equations.

PhaseChange: Determine phase change within soil/snow layers.

Hydrology2: Main subroutine to determine soil/snow hydrology.

SnowWater: Determine the change of snow mass and the snow water.

SurfaceRunoff: Determine surface runoff.

Infiltration: Determine infiltration into surface soil layer (minus the evaporation).

SoilWater: Determine soil moisture.

Drainage: Determine subsurface runoff.

SnowCompaction: Determine natural compaction and metamorphosis of snow.

CombineSnowLayers: Combine thin snow elements.

Combo: Combine two snow elements in terms of temperature, liquid water and ice contents, and layer thickess.

DivideSnowLayers: Divide thick snow elements.

WetIceHydrology: Calculate hydrology for ice and wetland. Maintains a constant water volume for wetlands and ice.

Hydrology_Lake: Determine fate of snow on lake. Force constant lake volume.

SnowAge: Determine age of snow for albedo calculations.

BalanceCheck: Error checks for energy and water balance.

histUpdate: Accumulate history fields over history time interval.

Rtmriverflux: Route surface and subsurface runoff into rivers for RTM river routing model.

Rtm: RTM river routing model.

histHandler: Main history file handler. This code 1) increments field accumulation counters at every time step and determines if next time step is beginning of history interval 2) at the end of a history interval, increments the current time sample counter, opens a new history file if needed, writes history data to current history file and resets field accumulation counters to zero and 3) when a history file is full, or at the last time step of the simulation, closes history file and disposes to mass store (only if file is open), resets time sample counter to zero (only if file is full) and increments file counter by one (only if file is full)

restwrt: Write binary restart files.

inicwrt: Write netCDF initial files.

2 Modifying CLM2.0 Input and Output Files

2.1 Modifying Surface Data

A list of high resolution "raw" datasets is provided with the CLM2.0 distribution. These raw datasets are only needed if a raw surface dataset is to be created at runtime (see CLM2.0 User's Guide). The following is a list of these datasets:

- mksrf_pft.nc: raw vegetation type dataset
- mksrf_soitex.10level.nc: raw soil texture dataset
- mksrf_soicol_clm2.nc: raw soil color dataset
- mksrf_lanwat.nc: raw inland water dataset
- mksrf_urban.nc: raw urban dataset
- mksrf_glacier.nc: raw glacier dataset
- mksrf_lai.nc: raw leaf and stem area index, canopy top and bottom height dataset

The user has the option of replacing any of these raw datasets, as long as the structure of the netCDF files is preserved. A series of fortran routines to convert ascii to raw netCDF data is provided in the directory, /tools/convert_ascii. Alternatively, the user may wish to modify the model surface data directly as long as the original netCDF file structure is preserved.

2.2 History fields

To output a history field which has not been included in the CLM2.0 code, the user must do the following:

Step 1. Add the appropriate initialization call for the required history field in subroutine **histlst**. For example, the following lines could be added in **histlst**:

```
call histfldini(nflds, 'NEWFLD ', 'UNITS', nsing, naver,'FIELD DESC', .true., histfld)
```

- where NEWFLD is the new field name.

- UNITS is the new field units.
- the level structure of the new history field is either "nsing" (for a single level field) or "nsoil" (for a multi-level "soil" field).
- the type of time averaging desired is set to either "naver" (average), "ninst" (instantaneous), "nmini" (minimum) or "nmaxi" (maximum)
- FIELD DESC is the new field description
- the field is specified to be active or non-active by default [.true., .false.] If value is .true. the field is active by default (ie, written to history).

Step 2. In subroutine **histUpdate**, add the following lines (using the following only as an example):

```
!$OMP PARALLEL DO PRIVATE (K)
do k = begpatch,endpatch
tmpslev(k) = call histslf('NEWFLD', tmpslev)
end do
```

where tmpslev and histslf are used for adding a new single-level field (whereas tmpmlev and histmlf are used for adding a new multi-level field). If adding a multi level field, the call to histmlf passes an additional variable denoting the number of levels (nlevsoi or nlevlak). It may also be necesasry to increase the maximum allowable number of single level fields. This can be done by increasing the values of the parameters **max_slevflds** or **max_mlevflds** in routine src/main/clm_varpar.F90.

2.3 Initial datasets

Initial datasets are created by the land model periodically (for details, see the CLM2.0 User's Guide). Initial datasets are netCDF files containing instantaneous variable data. Since certain model variables take relatively long to spin up (e.g. soil and snow related variables), using an initial dataset ensures faster spin up of the model than using arbitrary initialization. The variables stored in initial dataset files are dimensioned by number of landpoints (e.g., numland = 1956 for a T31 global resolution) and the maximum number of subgrid patches in a land gridcell (maxpatch = 8). If a variable has multiple vertical levels, an additional dimension (e.g. nlevsoi = 10) is used.

Changing an initial dataset should not be necessary most of the time. However, if new variables must be added to the initial dataset, the user should refer to subroutines **inicwrt** and **inicrd** in the file **src/main/inicFileMod.F90**. The user should start in subroutine inicwrt by defining the new variable as a single or multi-level field (exactly as is done for existing variables). In the same subroutine the user should add a few lines to write out the new variable (again as done for existing variables). Subsequently, the user should add a similar group of lines to subroutine **inicrd**, for the variable to be recognized by the code at initialization. Comments in the code are clear for the user to identify where variables are defined, where they are written, and where they are read.

An important distinction between initial and restart files is that initial files have a self-describing format (since they are netCDF), and therefore may be used by future versions of the model to initialize a simulation. Restart files (see below), on the other hand, may change frequently during model development and may be difficult to use with different versions of the code.

2.4 Restart files

Restart files contain instantaneous binary data for a set of time-dependent variables. The data is stored in binary format to ensure bit-for-bit agreement with a terminated run which was subsequently restarted.

Restart files contain a version number to ensure that the code being used is compatible with the restart file being read. If the user modifies the contents of the restart file, the version number should be incremented.

Restart files should be used to extend previously started simulations or to branch from one simulation to another in which history file namelist variables have been modified. For a branch run, the user must change the simulation's case name.

The criterion for a "successful restart" is that the model has available exactly the same information upon restart that it would have had if it had not stopped. Restart files may need to be modified during model development to include new time-dependent variables. A simple rule is that if a run produces even slightly different answers when restarted compared to when left uninterrupted, then certain variables are missing from the restart file.

Restart files are written in subroutine **restwrt** and read in subroutine **restrd** (which are contained in file src/main/restFileMod.F90. Subroutine **restrd** reads the data in exactly the same order as it was written out and the simulation continues as though uninterrupted. If a user modifies the code such that it becomes necessary to add a new variable to the restart file, separate entries must be made in subroutines **restwrt** and **restrd**. As an example, if the following field were to be added the the restart file, the following two entries would have to be made:

```
restwrt:
buf1d(begpatch:endpatch) = clm(begpatch:endpatch)%newvar
call wrtout (nio, buf1d)

restrd:
call readin (nio, buf1d)
clm(begpatch:endpatch)%newvar = buf1d(begpatch:endpatch)
```

The user must make sure to place the former two lines in exactly the same order with respect to existing variables as the latter two lines. For example, variable fwet is between variables t_grnd and tlai in both subroutines **restrd** and **restwrt**.

3 Error codes

Error codes exist in the model to warn the user of inconsistencies and errors. The model is released to the community as a package which should not trigger such inconsistencies or errors. However, the user may change the code or modify the input files in a way which triggers such a message. The user may even simply operate the model on a platform which brings up an error message. In many cases these error codes are followed by a stoprun in order to alert the user of a problem which will render the simulation useless or will cause it to fail later.

In general, when an error message appears, the user must switch gears from production to debugging mode. The error message may make the corrective measures obvious. If not, the user will need to find the line in the code which wrote the error message and gradually back track till the cause for the error is found. The old fashioned placement of write statements in the code or, if available, debugging software may help understand the problem. When using debugging software, it is useful to change DEBUG to .true. in the jobscript.

3.1 Energy and water balance errors

To make sure that the model conserves energy and mass, energy and water imbalances appearing in a simulation will trigger errors and halt the simulation,

To accomplish energy conservation, all energy inputs at the land surface are reflected or absorbed. Absorbed energy may return to the atmosphere as sensible heat or may be emitted as terrestrial radiation, stored as soil heat, used to evaporate or transpire water, or used to melt snow and ice. The error check for the energy balance is in subroutine src/main/BalanceCheck.F90.

Similarly, water incident on the land will evaporate, transpire through leaf stomata, run off and drain from the soil, or be stored in the soil and snow pack. The error check for the water balance is in subroutine BalanceCheck.

Water which runs off or drains from the soil fills river chanels and flows downstream. At every model time step, the global sum of water which runs off and drains from the soil into the rivers must equal the global sum of change in river volume. This is because the global sum of water flowing out of grid cells must cancel the global sum of water flowing into grid cells from upstream. This error check is in subroutine src/riverroute/RtmMod.F90.