CPL7 User's Guide

Tony Craig
NCAR

CPL7 User's Guide

by Tony Craig

Table of Contents

Chapter 1. CPL7 Introduction

How to Use This Guide

This guide covers the CCSM4 top level driver implementation as well as the coupler component within the system. The driver runs on all hardware processors and basically runs the top level instructions and executes the driver time loop. The coupler is a component of the CCSM4 system that is run from within the driver. It can be run on a subset of the total processors, and carries out mapping (interpolation), merging, diagnostics, and other calculations. The name cpl7 refers to the source code associated with both the driver aspects and the coupler component aspects. cpl7 code is located in the CCSM4 source tree under models/drv/ and the main program of CCSM4 is models/drv/driver/ccsm_driver.F90.

This document provides a general overview of the cpl7 design. Then specific implementation issues are discussed individually. Finally, there is a section summarizing all of the cpl7 namelist input. This document is written primarily to help users understand the inputs and controls within the cpl7 system, but to also provide some background about the associated implementation.

Chapter 2. CPL7 User Guide

General Overview

CCSM4 Driver/Coupler (Updated to drvseq3_1_20)

The CCSM4 Version of the Community Climate System Model has some significant changes compared to previous versions. In particular, CCSM4 is NOT run via multiple executables in a concurrent only processor layout. There is now a top level driver and components are called via standard init, run, and finalize methods. Components can be run sequentially, concurrently, or in some mixed sequential/concurrent layout on processors. A coupler component that runs on a subset of the total processors still exists in the system as part of the driver. The driver runs on all processors and the coupler functions (mapping, merging, flux calculations, and diagnostics) runs on a user defined subset of the total processors. The processor layout is specified at runtime via namelist inputs.

While the processor layout is relatively flexible and components can be run sequentially or concurrently, the sequencing of the science in the driver is fixed and independent of the processor layout. So changing the processor layout only changes the performance of the simulation system.

Like all components in CCSM, the driver scripting system is setup such that a component template file (models/drv/bld/cpl.template) is run by the ccsm configure script, and it generates partially resolved cpl.buildnml.csh and cpl.buildexe.csh scripts. The CCSM configure script also generates case .build, .run, and .clean_build scripts. See the CCSM user guide for more detailed information about this process. But briefly, the process is as follows,

```
- run create_newcase to generate a case directory.
- edit env_conf.xml and env_mach_pes.xml and then run configure -case.  this
  runs the coupler cpl.template file among other things.  the configure script
  generates the .build, .run, and .clean_build scripts and
  the partially resolved buildnml and buildexe scripts in the
  Buildconf directory.
- edit env_build.xml and run the .build script.
- edit env_run.xml and submit the .run script.
```

Design Discussion

Overview

In CCSM3/cpl6, all components ran as separate executables and there was no concept of a top-level driver. The components started independently and then communicated to the coupler at regular intervals via send and receive methods called directly from within each component. The coupler acted as a central hub coordinating data exchange, implicitly managing lags and sequencing, and executing coupler operations such as mapping (interpolation) and merging.

CCSM4/cpl7 is now built as a single executable with a single high-level driver. The driver runs on all processors and handles coupler sequencing, model concurrency, and communication of data between components. The driver calls all model components via common and standard interfaces. The driver also directly calls coupler methods for mapping (interpolation), rearranging, merging, an atmosphere/ocean flux calculation, and diagnostics. In CCSM4, the model components and the coupler methods can run on subsets of all the processors. In some sense, the cpl6 sequencing and hub attributes have been migrated to the driver level while the cpl6 coupler operations like mapping and merging are being done as if a separate coupler component existed in CCSM4. In other words, cpl7 consists of a driver that controls the top level sequencing, the processor decomposition, and communication to components through subroutine calls while coupler operations such as mapping and merging are running under the driver on a subset of processors as if there were a unique coupler model component.

CCSM4 consists of both data and active components models. In general, an active component both needs data from and provides data to the coupler while data models generally read data from I/O and then just provide data to the coupler. In CCSM4, like CCSM3, the atmosphere, land, and sea ice models are always tightly coupled to better resolve the diurnal cycle. This coupling is typically half-hourly, although at higher resolutions, coupling can be more frequent. The ocean model coupling is typically once or a few times per day. The diurnal cycle of ocean surface albedo is computed in the coupler for use by the atmosphere model. The looser ocean coupling frequency means the ocean forcing and response is lagged in the system. There is an option in CCSM4 to run the ocean tightly coupled without any lags, but this is normally used only when running with data ocean components.

Depending on the resolution, hardware, run length and physics, a CCSM4 run can take several hours to several months of wall time to complete. Runs are typically decades or centuries long, and the model typically runs between 1 and 50 model years per wall clock day. CCSM4 has exact restart capability and the model is typically run in individual one year or multi-year chunks. CCSM4 has automatic resubmission and automatic data archiving capability.

Sequencing and Concurrency

In CCSM4, the component processor layouts and MPI communicators are derived from namelist input. At the present time, there are seven (7) basic processor groups in CCSM. These are associated with the atmosphere, land, ocean, sea ice, land ice, coupler, and global groups, although others could be easily added later. Each of the seven processor groups can be distinct, but that is not a requirement of the system. A user can overlap processor groups relatively arbitrarily. If all processors sets overlap each other in at least one processor, then the model runs sequentially. If all processor sets are distinct, the model runs concurrently. The processor sets for each component group are described via 3 basic scalar parameters at the present time; the number of mpi tasks, the number of openmp threads per mpi task, and the global mpi task rank of the root mpi task for that group. For example, a layout where the number of mpi tasks is 8, the number of threads per mpi task is 4, and the root mpi task is 16 would create a processor group that consisted of 32 hardware processors, starting on global mpi task number 16 and it would contain 8 mpi tasks. The global group would have at least 24 tasks and at least 48 hardware processors. The driver derives all MPI communicators at initialization and passes them to the component models for use.

As mentioned above, there are two issues related to whether the component models run concurrently. The first is whether unique chunks of work are running on distinct processor sets. The second is the sequencing of this work in the driver. As much as possible, the CCSM4 driver sequencing has been implemented to maximize the potential amount of concurrency of work between different components. Ideally, in a single coupling step, the forcing for all models would be computed first, the models could then all run concurrently, and then the driver would advance. However, scientific requirements such as the coordination of surface albedo and atmosphere radiation computations as well as general computational stability issues in CCSM4 prevent this ideal implementation. Figure 1 shows the maximum amount of concurrency supported by the current CCSM4 driver implementation for a fully active system. In practice, the scientific constraints mean the active atmosphere model cannot run concurrently with the land and sea-ice models. Again, figure 1 does not necessarily represent the optimum processor layout for performance for any configuration, but it provides a practical limit to the amount of concurrency in the system due to scientific constraints. With CCSM4, results are bit-for-bit identical regardless of the component sequencing because the scientific lags are fixed in CCSM4 by the implementation, not the processor layout.

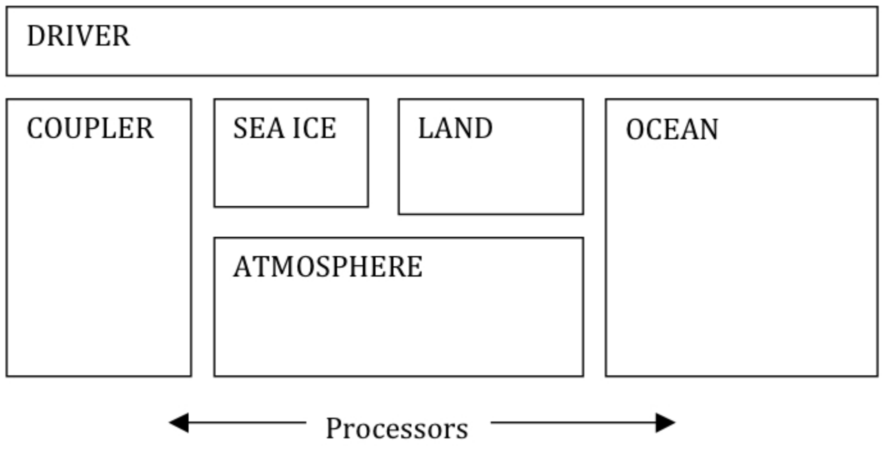

Figure 1: Maximum potential processor concurrency designed into CCSM4 to support scientific requirements and stability.

Relative to CCSM3, there is a loss of concurrency in CCSM4 because they are coupled in fundamentally different ways. In CCSM4, the model run methods are called from the driver and the coupling from the overall system perspective looks like "send to component, run component, receive from component". In CCSM3, the coupling was done via calls inside each component and the data send and receive could be interleaved with work such that the model run method appeared to have two phases, one between the coupling send and receive, the other between the receive and send. That allowed greater concurrency in CCSM3, but was designed for concurrent-only coupling. In CCSM4, we chose not to implement multiple run phases in order to simplify the model and provide an opportunity for greater interoperability. This choice was made with the full understanding that there would be some potential loss of model concurrency in specific cases. Designers felt the additional flexibility of allowing models to run in a mixed sequential/concurrent system would likely overcome any performance degradation associated with a loss of concurrency.

Component Interfaces

The standard CCSM4 component model interfaces are based upon the ESMF design. Each component provides an init, run, and finalize method with consistent arguments. The CCSM4 component interface arguments currently consist of Fortran and MCT datatypes, but an alternative ESMF version is also available. The physical coupling fields are passed through the interfaces in the init, run, and finalize phases. As part of initialization, an MPI communicator is passed from the driver to the component, and grid and decomposition information is passed from the component back to the driver. The driver/coupler acquires all information about resolution, configurations, and processor layout at run-time from either namelist or from communication with components.

Initialization of the system in CCSM4 is relatively straight-forward. First, the seven MPI communicators are computed in the driver. Then the atmosphere, land, ocean, sea ice, and land ice models' initialization methods are called on the appropriate processor sets, and mpi communicator is sent, and grid and decomposition information are passed back to the driver. Once the driver has all the grid and decomposition information from the components, various rearrangers and mappers are initialized that will move data between processors, decompositions, and grids as needed at the driver level. No distinction is made in the coupler implementation for sequential versus concurrent execution. In general, even for cases where two components have identical grids and processor layouts, often their decomposition is different for performance reasons. In cases where the grid, decomposition, and processor layout are identical between components, the mapping or rearranging operation will degenerate to a local data copy.

The interface to the components' run method consists of two distinct bundles of fields. One is the data sent to force the model. The second is data received from the model for coupling to other components. The run interface also contains a clock that specifies the current time and the run length for the model. These interfaces follow the ESMF design principles.

MCT, The Model Coupling Toolkit

MCT was a critical piece of software in the CCSM3 cpl6 coupler. In the updated CCSM4 cpl7 coupler, the MCT attribute_vector, global_segmap, and general_grid datatypes have been adopted at the highest levels of the driver, and they are used directly in the component init, run, and finalize interfaces. In addition, MCT is used for all data rearranging and mapping (interpolation). The clock used by CCSM4 at the driver level is based on the ESMF specification. Mapping weights are still generated off-line using the SCRIP package as a preprocessing step. They are read into CCSM4 using a subroutine that reads and distributes the mapping weights in reasonably small chunks to minimize the memory footprint. Development of the CCSM4 cpl7 coupler not only relies on MCT, but MCT developers contributed significantly to the design and implementation of the cpl7 driver. Development of both the cpl6 and cpl7 coupler has resulted from a particularly strong and close collaboration between NCAR and the Department of Energy Argonne National Lab.

Memory, Parallel IO, and Performance

CCSM4 is targeting much higher resolutions than any previous CCSM coupled model. In order to facilitate scientific climate model exploration and development at these scales, the technology had to be put in place first to enable testing of this case. Efforts have been made to reduce the memory footprint and improve memory scaling in all components with a target of being able to run the fully coupled system at one tenth (0.1) degree resolution globally on tens-of-thousands of processors with each processor having as little as 512 Mb of memory. This target limits the number of global arrays that can be allocated on any processor to just a few, at most, at any time. The memory limitations have imposed new constraints on component model initialization, and significant refactoring has been required in some models' initialitialization to reduce the amount of global memory used. In addition, all components have implemented I/O that supports reading and writing of only one (1) global horizontal array at a time through a master processor.

Development of PIO, a parallel I/O library based on netcdf, pnetcdf, and MPI-IO is well underway within the CCSM community to improve I/O performance and memory usage in the model. Most model components are currently using the pio software for I/O, and use of PIO has allowed testing of CCSM4 at high resolutions that were previously memory limited.

Scaling to tens-of-thousands of processors requires reasonable performance scaling of the models, and all components have worked at improving scaling via changes to algorithms, infrastructure, or decompositions. In particular, decompositions using shared memory blocking, space filling curves, and all three spatial dimensions have been implemented to varying degrees in all components to increase parallelization and improve scalability.

In practice, CCSM4 performance, load balance, and scalability are limited as a result of the size, complexity, and multiple model character of the system. Within the system, each component has its own scaling characteristics. In particular, each has processor count "sweet-spots" where the individual component model performs particularly well. This might occur within a component because of internal load balance, decomposition capabilities, communication patterns, or cache usage. Second, component performance can vary over the length of the model run. This occurs because of seasonal variability of the cost of physics in models, changes in performance during an adjustment (spin-up) phase, and temporal variability in calling certain model operations like radiation, dynamics, or I/O. Third, the hardware or batch queueing system might have some constraints on the total number of processors that are available. For instance, on 16 or 32 way shared memory node, a user is typically charged based on node usage, not processor usage. So there is no cost savings running on 40 processors versus 64 processors on a 32-way node system. As a result of all of these issues, load-balancing CCSM4 perfectly is generally not possible. But to a large degree, if one accepts the limitations, a load balance configuration with acceptable idle-time and reasonably good throughput is nearly always possible to configure. CCSM4 has significantly increased the flexibility of the possible processor layouts, and this has resulted in better load balance configurations in general.

Load-balancing CCSM4 requires a number of considerations such as which components are run, their absolute resolution, and their relative resolution; cost, scaling and processor count sweet-spots for each component; and internal load imbalance within a component. It is often best to load balance the system with all significant run-time I/O turned off because this occurs very infrequently (typically one timestep per month in CCSM4), is best treated as a separate cost, and can bias interpretation of the overall model load balance. Also, the use of OpenMP threading in some or all of the system is dependent on the hardware/OS support as well as whether the system supports running all MPI and mixed MPI/OpenMP on overlapping processors for different components. Finally, should the components run sequentially, concurrently, or some combination of the two. Typically, a series of short test runs is done with the desired production configuration to establish a reasonable load balance setup for the production job. CCSM4 provides some post-run analysis of the performance and load balance of the system to assist users in improving the processor layouts.

Chapter 3. CPL7 Implementation

Time Management

Driver Clocks

The CCSM4 driver manages the main clock in the system. That clock advances at the shortest coupling period and uses alarms to trigger component coupling and other events. In addition, the driver maintains a clock that is associated with each component. The driver's component clocks have a timestep associated with the coupling period of that component. The main driver clock and the component clocks in the driver advance in a coordinated manor and are always synchronized. The advancement of time is managed as follows in the main run loop. First, the main driver clock advances one timestep and the component clocks are advanced in a synchronous fashion. The clock time represents the time at the end of the next model timestep. Alarms may be triggered at that timestep to call the the atmosphere, land, sea ice, land ice, or ocean run methods. If a component run alarm is triggered, the run method is called and the driver passes that component's clock to that component. The component clock contains information about the length of the next component integration and the expected time of the component at the end of the integration period.

Generally, the component models have indepedent time management software. When a component run method is called, the component must advance the proper period and also check that their internal clock is consistent with the coupling clock before returning to the driver. The clock passed to the component by the driver contains this information. Component models are also responsible for making sure the coupling period is consistent with their internal timestep. History files are managed independently by each component, but restart files are coordinated by the driver (see the Section called *History and Restart Files*).

The clocks in CCSM4 are based on the ESMF clock datatype are are supported in software by either an official ESMF library or by software included in CCSM called esmf_wrf_timemgr. The esmf_wrf_timemgr software is a much simplified Fortran implementation of a subset of the ESMF time manager interfaces.

The Driver Time Loop

The driver time loop is hardwired to sequence the component models in a specific way to meet scientific requirements and to otherwise provide the maximum amount of potential concurrency of work. The results of the model integration are not dependent on the processor layout of the components.

In addition, the driver is currently configured to couple the atmosphere, land, and sea ice models using the same coupling frequency while the ocean model can be coupled at the same or at a lower frequency. To support this feature, the driver does temporal averaging of coupling inputs to the ocean and the driver also computes the surface ocean albedo at the higher coupling frequency. There is no averaging of coupling fields for other component coupling interactions and the land and sea ice models' surface albedos are computed inside those components. Averaging functionality could be added to the driver to support alternative relative coupling schemes in the future if desired with the additional caveat that the interaction between the surface albedo computation in each component and the atmospheric radiation calculation have to be carefully considered. In addition, some other features may need to be extended to support other coupling schemes and still allow model concurrency.

The coupler processors (pes) handle the interaction of data between components, so there are separate tasks associated with deriving fields on the coupler pes, transfering data to and from the coupler pes and other components, and then running the component models on their processors. The driver time loop is sequenced as follows,

```
The driver clock is advanced first.
Initial data is computed on the coupler pes.
Ocean data is rearranged from the coupler to the ocean pes.
Land data is rearranged from the coupler to the land pes.
Ice data is rearranged from the coupler to the ice pes.
The ocean model is run.
The ice model is run.
The land model is run.
The ocean inputs are accumulated, and the atmosphere/ocean fluxes are
  computed on the coupler pes based on the results from the previous
  coupled timestep.
Land data is rearranged from the land pes to the coupler pes.
Ice data is rearranged from the ice pes to the coupler pes.
Atmospheric forcing data is computed on the coupler pes.
Atmospheric data is rearranged from the coupler pes to the atmosphere pes.
The atmosphere model is run.
Atmospheric data is rearranged from the atmosphere pes to the coupler pes.
Ocean data is rearranged from the ocean pes to the coupler pes.
The loop returns
```

Within this loop, as much as possible, coupler work associated with mapping data, merging fields, diagnosing, applying area corrections, and computing fluxes is overlapped with component work. The land ice model interaction has not been discussed in the above context because the coupling sequencing of that component is still being developed, but the land ice model is likely to be coupled at much longer periods that the current components.

The driver sequencing in CCSM4 has been developed over nearly two decades, and it plays a critical role in conserving mass and heat, minimizing lags, and providing stability in the system. The above description is consistent with the concurrency limitations described here. Just to reiterate, the land and sea ice models will always run before the atmospheric model, and the coupler and ocean models are able to run concurrently with all other components. The coupling between the atmosphere, land, sea ice, and atmosphere/ocean flux computation incurs no lags but the coupling to the ocean state is lagged by one ocean coupling period in the system. Mass and heat are conserved in the system with more description here.

It is possible to reduce the ocean lag in the system. There is a namelist variable, ocean_tight_coupling, that moves the step where ocean data is rearranged from the ocean pes to the coupler pes from the end of the loop to before the atmosphere/ocean flux computation. If ocean_tight_coupling is set to true, then the ocean lag is reduced by one atmosphere coupling period, but the ability of the ocean model to run concurrently with the atmosphere model is also reduced or eliminated. This flag is most useful when the ocean coupling frequency matches the other components.

Coupling Frequency

In the current implementation, the coupling period must be identical for the atmosphere, sea ice, and land components. The ocean coupling period can be the same or greater. All coupling periods must be multiple integers of the smallest coupling period and should evenly divide a day of time.

The coupling periods are set in the driver namelist for each component via variables called something like atm_cpl_dt and atm_cpl_offset. The units of these inputs are seconds. The coupler template file derives these values from CCSM4 script variable names like ATM_NCPL which is the coupling frequency per day. The *_cpl_dt input specifies the coupling period in seconds and the *_cpl_offset input specifies the temporal offset of the coupling time relative to initial time. An example of an offset might be a component that couples every six hours. That would normally be on the 6th, 12th, 18th, and 24th hour of every day. An offset of 3600 seconds would change the coupling to the 1st, 7th, 13th, and 19th hour of every day. The offsets cannot be larger than the coupling period and the sign of the offsets is such that a positive offset shifts the alarm time forward by that number of seconds. The offsets are of limited use right now because of the limitations of the relative coupling frequencies.

Offsets play an important role in supporting concurrency. There is an offset of the smallest coupling period automatically introduced in every coupling run alarm for each component clock. This is only mentioned because it is an important but subtle point of the implementation and changing the coupling offset could have an impact on concurrency performance. Without this explicit automatic offset, the component run alarms would trigger at the end of the coupling period. This is fine for components that are running at the shortest coupling period, but will limit the ability of models to run concurrently for models that couple at longer periods. What is really required for concurrency is that the run alarm be triggered as early as possible and that the data not be copied from that component to the coupler pes until the coupling period has ended. The detailed implementation of this feature is documented in the seq_timemgr_mod.F90 file and the impact of it for the ocean coupling is implemented in the ccsm_driver.F90 code via use of the ocnrun_alarm and ocnnext_alarm variables.

Grids

Standard Grid Configurations

The standard implementation for grids in CCSM has been that the atmosphere and land models are run on identical grids and the ocean and sea ice model are run on identical grids. The ocean model mask is used to derive a complementary mask for the land grid such that for any given combination of atmosphere/land and ocean/ice grids, there is a unique land mask. This approach for dealing with grids is still used a majority of the time in CCSM4. But there is a new capability, called trigrid, that allows the atmosphere and land grids to be unique. A typical grid in CCSM4 is named something like 1.9x2.5_gx1v6 which is the finite volume "2 degree" atmosphere/land grid matched with the gx1v6 "1 degree" ocean/ice grid. This also has a shortname of f19_g16. The "out-of-the-box" supported grids, compsets, and machines in CCSM4 are generated automatically by running

```
create_newcase -list
```

from the scripts directory.

Historically, the ocean grid has been the higher resolution grid in CCSM. While that is no longer always the case, the current implementation largely reflects that presumption. The atmosphere/ocean fluxes in the coupler are computed on the ocean grid. A new namelist input which is not yet validated called aoflux_grid will allow the user to specify the atmosphere/ocean flux computation grid in the coupler in the future. In addition, the mapping approach used in CCSM4 also reflects the presumption that the ocean is generally higher resolution. Fluxes are always mapped using a locally conservative area average methods to preserve conservation. However, states are mapped using bilinear interpolation from the atmosphere grid to the ocean grid to better preserve gradients, while they are mapped using a locally conservative area average approach from the ocean grid to the atmosphere grid. These choices are based on the presumption that the ocean grid is higher resolution.

There has always been an option that all grids (atmosphere, land, ocean, and ice) could be identical, and this is still supported. There are a couple of namelist variables, samegrid_ao, samegrid_al, and samegrid_ro that tell the coupler whether to expect that the following grids; atmosphere/ocean, atmosphere/land, and runoff/ocean respectively are identical. These are set automaticaly in the driver namelist depending on the grid chosen and impact mapping as well as domain checking.

Trigrid Configurations

A new feature in CCSM4 allows the atmosphere and land grids to be unique. One example in the CCSM4 scripts is the ne30np4_1.9x2.5_gx1v6 (ne30_f19_g16) grid. This grid configuration consists of the ne30np4 homme grid for the atmosphere, the "2 degree" grid for the land, and the "1 degree" grid for the ocean/ice. Note that the trigrid implementation has undergone a reasonable amount of testing, but as of April, 2010, it is still considered a relatively new feature and continues to undergo careful scrutiny.

The trigrid implementation introduces an ambiguity in the definition of the mask. This ambiguity is associated with an inability to define an absolutely consistent ocean/land mask across all grids in the system. A decision was made in CCSM4 to support the trigrid as follows. The land mask is defined on the atmosphere grid as the complement of the ocean mask mapped conservatively to the atmosphere grid. Then the land and ocean masks are exactly complementary on the atmosphere grid where conservative merging are critical. No precise land fraction needs to be defined in the land grid. The only requirement is that the land model compute data on a masked grid such that when mapped to the atmosphere grid, all atmosphere grid points that contain some fraction of land have valid values computed in the land model. There are an infinite number of land fraction masks that can accomplish this including a fraction field that is exactly one at every grid cell. In the land model, all land fraction masks produce internally conservative results. So there is no problem with conservation in the system with this approach with one caveat. The runoff model in the clm active land model is run on a unique half degree grid. In order to interpolate the land forcing data for the runoff grid, a land fraction mask that is consistent with the ocean mask in the system is required. To support the land to runoff mapping in clm, the land fraction on the atmosphere grid is conservatively mapped to the land grid in the coupler and sent to the land model for use. This field can then be used with the land fraction field from within clm to generated properly scaled runoff forcing which is also conservative.

Fractions

The component grid fractions in the coupler are defined and computed in CCSM4 in models/drv/driver/seq_frac_mct.F90. A slightly modified version of the notes from this file is pasted below. Just to clarify some of the terms. fractions_a, fractions_l, fractions_i, and fractions_o are the fractions on the atmosphere, land, ice, and ocean grids. afrac, lfrac, ifrac, and ofrac are the atmosphere, land, ice, and ocean fractions on those grids. so fractions_a(lfrac) is the land fraction on the atmosphere grid. lfrin in the land fraction defined in the land model. This can be different from lfrac because of the trigrid implementation. lfrac is the land fraction consistent with the ocean mask and lfrin is the land fraction in the land model. ifrad and ofrad are fractions at the last radiation timestep. These fractions preserve conservation of heat in the net shortwave calculation because the net shortwave calculation is one timestep behind the ice fraction evolution in the system. When the variable "dom" is mentioned below, that refers to a field sent from a component at initialization.

```
!  the fractions fields are now afrac, ifrac, ofrac, lfrac, and lfrin.
!    afrac = fraction of atm on a grid
!    lfrac = fraction of lnd on a grid
!    ifrac = fraction of ice on a grid
!    ofrac = fraction of ocn on a grid
!    lfrin = land fraction defined by the land model
!    ifrad = fraction of ocn on a grid at last radiation time
!    ofrad = fraction of ice on a grid at last radiation time
!      afrac, lfrac, ifrac, and ofrac are the self-consistent values in the
!      system.  lfrin is the fraction on the land grid and is allowed to
!      vary from the self-consistent value as descibed below.  ifrad
!      and ofrad are needed for the swnet calculation.
!  the fractions fields are defined for each grid in the fraction bundles as
!    needed as follows.
!    character(*),parameter :: fraclist_a = 'afrac:ifrac:ofrac:lfrac:lfrin'
!    character(*),parameter :: fraclist_o = 'afrac:ifrac:ofrac:ifrad:ofrad'
!    character(*),parameter :: fraclist_i = 'afrac:ifrac:ofrac'
!    character(*),parameter :: fraclist_l = 'afrac:lfrac:lfrin'
!    character(*),parameter :: fraclist_g = 'gfrac'
!
!  we assume ocean and ice are on the same grids, same masks
!  we assume ocn2atm and ice2atm are masked maps
!  we assume lnd2atm is a global map
!  we assume that the ice fraction evolves in time but that
!    the land model fraction does not.  the ocean fraction then
!    is just the complement of the ice fraction over the region
!    of the ocean/ice mask.
!  we assume that component domains are filled with the total
!    potential mask/fraction on that grid, but that the fractions
!    sent at run time are always the relative fraction covered.
!    for example, if an atm cell can be up to 50% covered in
!    ice and 50% land, then the ice domain should have a fraction
!    value of 0.5 at that grid cell.  at run time though, the ice
!    fraction will be between 0.0 and 1.0 meaning that grid cells
!    is covered with between 0.0 and 0.5 by ice.  the "relative" fractions
!    sent at run-time are corrected by the model to be total fractions
!    such that
!  in general, on every grid,
!              fractions_*(afrac) = 1.0
!              fractions_*(ifrac) + fractions_*(ofrac) + fractions_*(lfrac) = 1.0
!  where fractions_* are a bundle of fractions on a particular grid and
!    *frac (ie afrac) is the fraction of a particular component in the bundle.
!
!  the fractions are computed fundamentally as follows (although the
!    detailed implementation might be slightly different)
!  initialization (frac_init):
!    afrac is set on all grids
!      fractions_a(afrac) = 1.0
!      fractions_o(afrac) = mapa2o(fractions_a(afrac))
!      fractions_i(afrac) = mapa2i(fractions_a(afrac))
!      fractions_l(afrac) = mapa2l(fractions_a(afrac))
!    initially assume ifrac on all grids is zero
!      fractions_*(ifrac) = 0.0
!    fractions/masks provided by surface components
!      fractions_o(ofrac) = dom_o(frac)  ! ocean "mask"
!      fractions_l(lfrin) = dom_l(frac)  ! land model fraction
!    then mapped to the atm model
!      fractions_a(ofrac) = mapo2a(fractions_o(ofrac))
!      fractions_a(lfrin) = mapl2a(fractions_l(lfrin))
!    and a few things are then derived
!      fractions_a(lfrac) = 1.0 - fractions_a(ofrac)
!        this is truncated to zero for very small values (< 0.001)
!        to attempt to preserve non-land gridcells.
!      fractions_l(lfrac) = mapa2l(fractions_a(lfrac))
!    one final term is computed
!      dom_a(ascale) = fractions_a(lfrac)/fractions_a(lfrin)
!      dom_l(ascale) = mapa2l(dom_a(ascale))
!        these are used to correct land fluxes in budgets and lnd2rtm coupling
!        and are particularly important when the land model is running on
!        a different grid than the atm model.  in the old system, this term
!        was treated as effectively 1.0 since there was always a check that
!        fractions_a(lfrac) ~ fractions_a(lfrin), namely that the land model
!        provided a land frac that complemented the ocean grid.  this is
!        no longer a requirement in this new system and as a result, the
!        ascale term can be thought of as a rescaling of the land fractions
!        in the land model to be exactly complementary to the ocean model
!        on whatever grid it may be running.
!  run-time (frac_set):
!    update fractions on ice grid
!      fractions_i(ifrac) = i2x_i(Si_ifrac)  ! ice frac from ice model
!      fractions_i(ofrac) = 1.0 - fractions_i(ifrac)
!        note: the relative fractions are corrected to total fractions
!      fractions_o(ifrac) = mapi2o(fractions_i(ifrac))
!      fractions_o(ofrac) = mapi2o(fractions_i(ofrac))
!      fractions_a(ifrac) = mapi2a(fractions_i(ifrac))
!      fractions_a(ofrac) = mapi2a(fractions_i(ofrac))
!
!  fractions used in merging are as follows
!    mrg_x2a uses fractions_a(lfrac,ofrac,ifrac)
!    mrg_x2o needs to use fractions_o(ofrac,ifrac) normalized to one
!      normalization happens in mrg routine
!
!  fraction corrections in mapping are as follows
!    mapo2a uses *fractions_o(ofrac) and /fractions_a(ofrac)
!    mapi2a uses *fractions_i(ifrac) and /fractions_a(ifrac)
!    mapl2a uses *fractions_l(lfrin) and /fractions_a(lfrin)
!    mapa2* should use *fractions_a(afrac) and /fractions_*(afrac) but this
!      has been defered since the ratio always close to 1.0
!
!  budgets use the standard afrac, ofrac, ifrac, and lfrac to compute
!    quantities except in the land budget which uses lfrin multiplied
!    by the scale factor, dom_l(ascale) to compute budgets.
!
!  fraction and domain checks
!    initialization:
!      dom_i = mapo2i(dom_o)  ! lat, lon, mask, area
!      where fractions_a(lfrac) > 0.0, fractions_a(lfrin) is also > 0.0
!         this ensures the land will provide data everywhere the atm needs it
!         and allows the land frac to be subtlely different from the
!         land fraction specified in the atm.
!      dom_a = mapl2a(dom_l)  ! if atm/lnd same grids
!      dom_a = mapo2a(dom_o)  ! if atm/ocn same grids
!      dom_a = mapi2a(dom_i)  ! if atm/ocn same grids
!      0.0-eps < fractions_*(*) < 1.0+eps
!      fractions_l(lfrin) = fractions_l(lfrac)
!        only if atm/lnd same grids (but this is not formally required)
!        this is needed until dom_l(ascale) is sent to the land model
!        as an additional field for use in l2r mapping.
!    run time:
!      fractions_a(lfrac) + fractions_a(ofrac) + fractions_a(ifrac) ~ 1.0
!      0.0-eps < fractions_*(*) < 1.0+eps
```

Domain Checking

Domain checking is a very important initialization step in the system. The domain checking verifies that the longitudes, latitudes, areas, masks, and fractions of different grids are consistent with each other in a way that is required by the CCSM4 implementation. The subroutine that carries out domain checking is in models/drv/driver/seq_domain_mct.F90 and is called seq_domain_check_mct. Tolerances for checking the domains can be set in the drv_in driver namelist via the namelist variables, eps_frac, eps_amask, eps_agrid, eps_aarea, eps_omask, eps_ogrid, and eps_oarea. These values are derived in the coupler namelist from the script env variables, EPS_FRAC, EPS_AMASK, EPS_AGRID, EPS_AAREA, EPS_OMASK, EPS_OGRID, and EPS_OAREA in the env_run.xml file. If an error is detected in the domain checking, the model will write an error message and abort.

The domain checking is dependent on the grids and in particular, the samegrid input namelist settings. But it basically does the following,

```
ocean/ice grid comparison:
  - verifies the grids are the same size
  - verifies the difference in longitudes and latitudes is less than eps_ogrid.
  - verifies the difference in masks is less than eps_omask
  - verifies the difference in areas is less than eps_oarea

atmosphere/land grid comparison (if samegrid_al):
  - verifies the grids are the same size
  - verifies the difference in longitudes and latitudes is less than eps_agrid.
  - verifies the difference in masks is less than eps_amask
  - verifies the difference in areas is less than eps_aarea

atmosphere/ocean grid comparison (if samegrid_ao):
  - verifies the grids are the same size
  - verifies the difference in longitudes and latitudes is less than eps_agrid.
  - verifies the difference in masks is less than eps_amask
  - verifies the difference in areas is less than eps_aarea

fractions
  - verifies that the land fraction on the atmosphere grid and the
    ocean fraction on the atmosphere grid add to one within a tolerance of
    eps_frac.
```

There are a number of subtle aspects in the domain checking like whether to check over masked grid cells, but these issues are less important than recognizing that errors in the domain checking should be treated seriously. It is easy to make the errors go away by changing the tolerances, but it is also easy to overlook a critical grid error that can impact conservation and consistency in a simulation.

Mapping (Interpolation)

Mapping files to support interpolation of fields between grids are computed offline. General, this is done using the SCRIP package, but any package that generates a mapping file of valid format can be used in CCSM. Several different mapping approaches are used in CCSM4. First, note that historically, the ocean grid has been the higher resolution grid in CCSM. While that is no longer always the case, the current implementation largely reflects that presumption. In general, mapping of fluxes is done using a locally conservative area average approach to preserve conservation. State fields are generally mapped using bilinear interpolation from the atmosphere grid to the ocean grid to better preserve gradients, but state fields are generally mapped using the conservative area average approach from the ocean grid to the atmosphere grid. But this is not a requirement of the system. The individual state and flux mapping files are specified at runtime using the seq_maps.rc input file, and any valid mapping file using any mapping approach can be specified in that input file.

The seq_maps.rc file contains information about the mapping files as well as the mapping type. There are currently two types of mapping implementations, "X" and "Y". The "X" mapping rearranges the source data to the destination grid decomposition and then a local mapping is done from the source to the destination grid on the destination decomposition. The "Y" mapping does a local mapping from the source grid to the destination grid on the source grid decomposition. That generates a partial sum of the destination values which are then rearranged to the destination decomposition and summed. Both options produce reasonable results, although they may differ in value by "roundoff" due to differences in order or operations. The type chosen impacts performance. In both implementations, the number of flops is basically identical. The difference is the communication. In the "X" type, the source grid is rearranged. In the "Y" type, the destination grid is rearranged. Since historically, the ocean grid is higher resolution than the atmosphere grid, "X" mapping is used for atmosphere to ocean/ice mapping and "Y" mapping is used from ocean/ice to atmosphere mapping to maximize mapping performance.

Mapping corrections are made in some cases in the polar region. In particular, the current bilinear and area conservative mapping approaches introduce relatively large errors in mapping vector fields around the pole. The current coupler corrects the interpolated surface wind velocity near the pole when mapping from the atmosphere to the ocean and ice grids. This only affects ocean and ice grid cells that are northward of the last latitude line of the atmospheric grid. The algorithm is contained in the file models/drv/driver/map_atmocn_mct.F90 and is only valid when the atmosphere grid is a longitude/latitude grid. This feature is generally on by default, but it can be turned off manually using the npfix namelist input in the drv_in file. This feature is turned off automatically by grid settings for grids in CCSM that are not longitude/latitude grids such as the homme dycore atmospheric grid.

The input mapping files are assumed to be valid for grids with masks of value zero or one where grids points with a mask of zero are never considered in the mapping. Well defined, locally conservative area mapping files as well as bilinear mapping files can be generated using this masked approach. However, there is another issue which is that a grid fraction in an active cell might actually change over time. This is not the case for land fraction in CCSM, but it is the case for relative ice and ocean fractions in CCSM. The ice fraction is constantly evolving in the system in general. To improve the accuracy of the ice and ocean mapping, the ocean/ice fields are scaled by the local fraction before mapping and unscaled by the mapped fraction after mapping. The easiest way to demonstate this is via an example. Consider a case where two ice cells of equal area underlie a single atmosphere cell completely. The mapping weight of each ice cell generated offline would be 0.5 in this case and if ice temperatures of -1.0 and -2.0 in the two cells respectively were mapped to the atmosphere grid, a resulting ice temperature on the atmosphere grid of -1.5 would result. Consider the case where one cell has an ice fraction of 0.3 and the other has a fraction of 0.5. Mapping the ice fraction to the atmospheric cell results in a value of 0.4. If the same temperatures are mapped in the same way, a temperature of -1.5 results which is reasonable, but not entirely accurate. Because of the relative ice fractions, the weight of the second cell should be greater than the weight of the first cell. Taking this into account properly results in a fraction weighted ice temperature of -1.625 in this example. This is the fraction correction that is carried out whenever ocean and ice fields are mapped to the atmosphere grid. Time varying fraction corrections are not required in other mappings to improve accuracy because their relative fractions remain static.

Area Correction of Fluxes

To improve conservation in the system, all fluxes sent to and received from components are corrected for the area differences between the components. There are many reasonable ways to compute an area of a grid cell, but they are not generally consistent. One assumption with respect to conservation of fluxes is that the area acting upon the flux is well defined. Differences in area calculations can result in differences of areas up to a few percent and if these are not corrected, will impact overall mass and heat conservation. In CCSM4, areas are extracted for each grid from the mapping files. In this implementation, it is assumed that the areas in all mapping files are computed reasonably and consistently for each grid and on different grids. Those mapping areas are used to correct the fluxes for each component by scaling the fluxes sent to and received by the component by the ratio of the mapping area and the component area. The areas from the components are provided to the coupler by the component at initialization. The minimum and maximum value of each area corrections is written to the coupler log file at initialization. One critical point is that if mapping files are generated by different tools offline and used in CCSM, an error could be introduced that is related to inconsistent areas provided by different mapping files.

Initialization

The CCSM initialization has been developed over the last two decades to meet the scientific goals, minimize the communication required, and ensure a consistent and well defined climate system. The order of operations is critical. The CCSM4 initialization is basically as follows,

```
The ccsm_pes namelist is read and mpi communicators are initialized.
The seq_infodata namelist is read and configuration settings are established.
The prof_inparm namelist is read and the timing tool is initialized.
The pio_inparm namelist is read and the driver IO is initialized.
The seq_timemgr namelist is read and the driver time manager and clocks
  are initialized.
The atmosphere init routine is called, the mpi communicator and clock are sent,
  and the atmosphere grid is returned.
The land init routine is called, the mpi communicator and clock are sent,
  and the land grid is returned.
The ocean init routine is called, the mpi communicator and clock are sent,
  and the ocean grid is returned.
The ice init routine is called, the mpi communicator and clock are sent,
  and the ice grid is returned.
The land ice init routine is called, the mpi communicator and clock are sent,
  and the land ice grid is returned.
The infodata buffer is synchronized across all processors.  This buffer
  contains many model configuration settings set by the driver but also
  sent from the components.
The mapping areas are initialized to the component areas on the component pes.
The atmosphere, land, runoff, ice, land ice, and ocean rearrangers
  are initialized.  These rearrangers move component data between the
  component pes and the coupler pes.
Initialize the coupler to component fields and zero them out.
The ocean accumulator is initialized and zeroed out.  This is used to
  accumulate and average the ocean input.
Datatypes associated with merging are allocated.
The atm2ocn, ocn2atm, ocn2ice, ice2ocn, ice2atm, rof2ocn, atm2lnd, and
  lnd2atm mapping weights and areas are read.
Component grids are checked using the domain checking method.
The flux area corrections are initialized on the component pes and applied
  to the initial fields sent by each component on the component pes.  Those
  initial fields are then rearranged to the coupler pes.
The fractions are initialized on the coupler pes.
The atmosphere/ocean flux computation is initialized and initial ocean albedos
  are computed on the coupler pes.
The land, ocean, and ice initial data is mapped to the atmosphere grid and
  merged to generate initial atmosphere forcing data.  This is really only
  needed for the albedo fields.
The initial atmosphere forcing data (albedos) is rearranged from the coupler
  pes to the atmosphere pes, and the area corrections are applied.
The atmosphere init method is called to run the second phase which initializes
  the atmosphere radiation based on the surface albedos sent.
The new atmosphere initial data is area corrected and rearranged to the
  coupler pes.
The budget diagnostics are zeroed out.
Initialization is complete.
```

MCT and ESMF Coupling Interfaces

CCSM4 supports both CCSM designed coupling interfaces based on MCT datatypes and ESMF coupling interfaces based on the ESMF design. In both cases, a top level driver calls init, run, and finalize methods for each gridded component. The primary and default method for running CCSM4 is with the MCT based interfaces and an ESMF library is not required in that case.

ESMF interfaces are supported via translation methods that are instantiated in the models/drv/shr_esmf directory. These methods translate between the datatypes in the MCT interfaces and the datatypes in the ESMF interfaces. In the current CCSM4 ESMF interface implementation, the top level driver still calls the init, run, and finalize component methods using the MCT interfaces. That interface does not match the ESMF interface provided by the gridded components. To facilitate translation, each component provides an additional layer where the MCT interfaces are translated to the ESMF interfaces and data is copied between datatypes on entry and exit of each method. The translation of MCT to ESMF datatypes and vice versa is supported via the shared source code in the models/drv/shr_esmf directory. In the future, the CCSM4 driver could be modified and the ESMF interfaces called directly thereby eliminating the extra translation layer.

All CCSM4 components support both MCT interfaces and ESMF interfaces at the top level independently. This is specifically implemented using two distinct directories of source code to separate the interfaces, so one or the other can be compiled with the component model. These directories exist in all components and are generally called cpl_mct and cpl_esmf. The cpl_mct directory contains the MCT coupling interface. The cpl_esmf directory contains the ESMF coupling interfaces as well as the additional translation layer. At the present time, these interfaces are maintained independently and modifications within any of the interface methods will likely require modifications to the alternative interface for consistency.

To use the ESMF interfaces, the ESMF version 4 release library must be installed locally and the USE_ESMF_LIB environment variable must be set to TRUE in the case env_build.xml file. In addition, the COMP_INTERFACE value in env_build.xml must be set to ESMF. When those variables are set, components compile the source code in cpl_esmf instead of cpl_mct and the ESMF library is linked to the CCSM executable. Support of the ESMF interfaces is a relatively new feature in CCSM4. Results using either interface are bit-for-bit identical for several configurations tested, and validation of the implementation is ongoing.

Driver Threading Control

OpenMP thread counts are controlled at three levels in CCSM4. The coarsest level is prior to launching the model in the CCSM run script. The environment variable OMP_NUM_THREADS is usually set to the largest value any mpi task will use in CCSM4. At a minimum, this will ensure threading is turned on to the maximum desired value in the run. The next level is during initialization in CCSM4. When the mpi communicators are initialized, the maximum number of threads per mpi task can be computed based on the ccsm_pes namelist input. At that point, there is an initial fortran call to the intrinsic, omp_set_num_threads. When that happens and if that call is successful, the number of threads will be set to the maximum needed in the system on an mpi task by task basis. Finally, there is the ability of CCSM to change the thread count per task as each component is individually called and as the model integrates through the driver run loop. In other words, for components that share the same hardware processor but have different threads per task, this feature allows those components to run with the exact value set by the user in the ccsm_pes namelist. This final level of thread control is turned off by default, but it can be turned on using the drv_threading namelist input. This fine control of threading is likely of limited use at this point given the current CCSM4 driver implementation.

The bit-for-bit (BFB) Flag

There is a bit-for-bit flag in the CCSM drv_in namelist called bfbflag. This flag modifies some coupler computations to preserve bit-for-bit results on different coupler processor counts. This flag has no impact on other components and their ability to generate bit-for-bit results on different pe counts. When this flag is set, all mappings become "X" types where the source data is rearranged to the destination processor and then local mapping is carried out. The order of operations of this mapping is independent of the pe count or decomposition of the grids. The other feature that is changed by the bfbflag is the global sum diagnostics. When the bfbflag is set to false, a partial sum is done on each processors and those partial sums are added together to form a global sum. This is generally not order of operations independent for different pe counts or decompositions. When the bfbflag is set, the global sums are computed by gathering the global field on the root processor and doing an ordered sum there.

History and Restart Files

In addition to log files, component models also produce history and restart files. History files are generally netcdf format and contain fields associated with the state of the model. History files are implemented and controlled independently in the component models, although support for monthly average history files is a standard output of most CCSM production runs. CCSM has a file naming standard for history files which includes the case names, component name, and model date.

All component models in CCSM must be able to stop in the middle of a run and then subsequently restart in a bit-for-bit fashion. For most models, this requires the writing of a restart file. The restart file can be any format, although netcdf has become relatively standard, and it should contain any scalars, fields, or information that is required to restart the component model in exactly the same state as when the restart was written and the model was stopped. The expectation in CCSM is that a restart of a model run will be bit-for-bit identical and this is regularly tested as part of CCSM development by running the model 10 days, writing a restart at the end of 5 days, and then restarting at day 5 and comparing the result with the 10 day run. Unlike history files, restart files must be coordinated across different components. The restart frequency is set in the driver time manager namelist and the driver triggers a restart alarm in clocks when a coordinated restart is requested. The components are required to check this alarm whenever they are called and to write a restart file at the end of the current coupling period. This method ensures all components are writing restart files at a consistent timestamp. The restart filenames are normally set in a generic rpointer file. The rpointer file evolves over the integration and keeps track of the current restart filenames. When a model is restarted, both the rpointer file and the actual restart file are generally required.

Many models are also able to restart accumulating history exactly files in the middle of an accumulation period, but this is not a current requirement in CCSM4. In production, the model is usually started and stopped on monthly boundaries so monthly average history files are produced cleanly. The run length of a CCSM4 production run is usually specified using the nmonths or nyears option and restart files are normally written only at the end of the run.

Budget Setup and Computation

Mass and heat are conserved in the coupler to several digits over centuries. Several steps have been taken to ensure this level of conservation, and these are described in other sections of the document. In addition, efforts have been made to make sure each component is internally conservative with respect to mass and heat.

The budgets can be turned on and off using the namelist variable do_budgets. The value of that namelist is set by the env variable, BUDGETS in env_run.xml. By default, the do_budgets flag is false and budgets are not generated. The CCSM4 coupler can diagnose the CCSM4 budget at several levels and over different periods. The periods are instantenous, daily average, monthly average, annual average, or since the start of the run. The budget output for each of these periods is controlled by the namelist input budget_inst, budget_daily, budget_month, budget_ann, budget_ltann, and budget_ltend. budget_ltann and budget_ltend are used to write the long term budget at either the end of every year or the end of every run. Other budgets are written at their period interval. The namelist input is an integer specifying what to write. The budget flags are controlled by env variables in env_run.xml named BUDGET_INST, BUDGET_DAILY, BUDGET_MONTHLY, BUDGET_ANNUAL, BUDGET_LONGTER_EOY, and BUDGET_LONGTERM_STOP respectively. Valid values are 0, 1, 2, or 3. If 0 is set, no budget data is written. The value 1 generates a net heat and water budget for each component, 2 adds a detailed heat and water budget for each component, and 3 adds a detailed heat and water budget of the different conmponents on the atmosphere grid. Normally values of 0 or 1 are specified. Values of 2 or 3 are generally used only when debugging problems involving conservation.

Chapter 4. CPL7 Namelist

Namelist Overview

In general, a user can modify the driver/coupler settings by editing either env variables or the buildnml files under Buildconf. Specifically, the driver/coupler namelist can be found in and will be generated by the script, $CASEROOT/Buildconf/cpl.buildnml.csh.

There are several namelists associated with the driver/coupler and they will be described below. Not all namelist variables are resolved in the buildnml script, but they all can be used. Below, a description of each namelist group and each namelist variable in each group will be presented including a summary of the type and defaults and whether any env variable is associated with the setting. The types are either char (character string), int (integer), r4 (real*4), r8 (real*8), or log (logical).

There are three important driver/coupler namelist input files (cpl_modelio.nml, drv_in, and seq_maps.rc) and each file has at least one namelist input group. The namelist groups are modelio, seq_infodata_inparm, seq_timemgr_inparm, ccsm_pes, prof_inparm, pio_inparm, and seq_maps. The files, namelist groups, and list of variables in each namelist group are

```
cpl_modelio.nml (input_filename)
  modelio (cpl_modelio.nml namelist)
    diri - obsolete
    diro - obsolete - obsolete
    logfile - logfile name

drv_in (input filename)
  seq_infodata_inparm (drv_in namelist)
    case_desc - case description
    case_name - case name
    start_type - start type
    model_version - model version
    username - current username
    hostname - hostname
    brnch_retain_casename - retain branch casename
    info_debug - debug level
    bfbflag - bit for bit flag
    drv_threading - turn on variable threading in driver
    samegrid_ao - atm and ocean samegrid flag
    samegrid_al - atm and land samegrid flag
    samegrid_ro - runoff and ocean samegrid flag
    eps_frac - domain checking fraction tolerance
    eps_amask - domain checking atm/lnd mask tolerance
    eps_agrid - domain checking atm/lnd grid tolerance
    eps_aarea - domain checking atm/lnd area tolerance
    eps_omask - domain checking ocn/ice mask tolerance
    eps_ogrid - domain checking ocn/ice grid tolerance
    eps_oarea - domain checking ocn/ice area tolerance
    shr_map_dopole - polar correction in shr_map
    npfix - north pole fix for vectors in a2o mapping
    flux_epbal - fresh water balance
    flux_albav - albedo averaging mode
    aoflux_grid - grid for atmocn flux calculation
    ocean_tight_coupling - tight ocean coupling flag
    perpetual - perpetual mode, untested
    perpetual_ymd, perpetual date, untested
    orb_iyear_AD - orbital year
    orb_obliq - orbital obliquity
```

orb_eccen - orbital eccentricity
orb_mvelp - orbital vernal equinox
atm_adiabatic - atm adiabatic mode
atm_ideal_phys - atm ideal physics mode
aqua_planet - aqua planet
single_column - single column mode
scmlon - single column longitude
scmlat - single column latitude
do_budgets - turn on budgets
budget_inst - instantaneous budget flag
budget_daily - daily budget flag
budget_month - monthly budget flag
budget_ann - annual budget flag
budget_ltann - long term annual budget flag
budget_ltend - long term end of run budget flag
cpl_cdf64 - turn on 64bit netcdf files in driver
restart_pfile - restart pointer file
restart_file - restart filename
timing_dir - timing output directory
tchkpt_dir - timing checkpoint directory
logFilePostFix - untested
outPathRoot - untested
histaux_a2x - driver aux history files for a2x fields
histaux_a2x3hr - driver aux history files for a2x 3 hour fields
histaux_a2x3hrp - driver aux history file for a2x 3 hour precip fields
histaux_a2x24hr - driver aux history file for a2x daily fields
histaux_l2x - driver aux history file for l2x fields
histaux_r2x - driver aux history file for r2x fields

seq_timemgr_inparm (drv_in namelist)
calendar - calendar type
start_ymd - start year/month/day
start_tod - start time of day
ref_ymd - reference year/month/day
ref_tod - reference time of day
curr_ymd - current year/month/day, obsolete
curr_tod - current time of day, obsolete
atm_cpl_dt - atm coupling period
ocn_cpl_dt - ocn coupling period
ice_cpl_dt - ice coupling period
lnd_cpl_dt - lnd coupling period
glc_cpl_dt - glc coupling period
atm_cpl_offset - atm coupling offset
lnd_cpl_offset - lnd coupling offset
ocn_cpl_offset - ocn coupling offset
ice_cpl_offset - ice coupling offset
glc_cpl_offset - glc coupling offset
stop_option - stop flag
stop_n - stop value
stop_ymd - stop date
stop_tod - stop time of day
restart_option - restart flag
restart_n - restart value
restart_ymd - restart date
end_restart - restart at end of run
history_option - driver snapshot history flag
history_n - driver snapshot history value
history_ymd - driver snapshot history date
histavg_option - driver average history flag
histavg_n - driver average history value

```
    histavg_ymd - driver average history date
    tprof_option - timing flag
    tprof_n - timing value
    tprof_ymd - timing date

  ccsm_pes (drv_in namelist)
    atm_ntasks - atm mpi tasks
    atm_rootpe - atm root task in global comm
    atm_pestride - atm mpi task stride
    atm_nthreads - atm threads per mpi task
    lnd_ntasks - lnd mpi tasks
    lnd_rootpe - lnd root task in global comm
    lnd_pestride - lnd mpi task stride
    lnd_nthreads - lnd threads per mpi task
    ice_ntasks - ice mpi tasks
    ice_rootpe - ice root task in global comm
    ice_pestride - ice mpi task stride
    ice_nthreads - ice threads per mpi task
    ocn_ntasks - ocn mpi tasks
    ocn_rootpe - ocn root task in global comm
    ocn_pestride - ocn mpi task stride
    ocn_nthreads - ocn threads per mpi task
    glc_ntasks - glc mpi tasks
    glc_rootpe - glc root task in global comm
    glc_pestride - glc mpi task stride
    glc_nthreads - glc threads per mpi task
    cpl_ntasks - cpl mpi tasks
    cpl_rootpe - cpl root task in global comm
    cpl_pestride - cpl mpi task stride
    cpl_nthreads - cpl threads per mpi task

  prof_inparm (drv_in namelist)
    profile_disable
    profile_barrier
    profile_single_file
    profile_global_stats
    profile_depth_limit
    profile_detail_limit
    profile_outpe_num
    profile_outpe_stride
    profile_timer
    profile_papi_enable

  pio_inparm (drv_in namelist)
    cpl_io_numtasks - number of io tasks in pio
    cpl_io_root - io task root in pio
    cpl_io_stride - stride of tasks in pio
    cpl_io_typename - io type in pio

seq_maps.rc (input filename)
  seq_maps (not a Fortran namelist)
    atm2ocnFmapname - atm to ocn flux mapping file
    atm2ocnFmaptype - atm to ocn flux mapping type
    atm2ocnSmapname - atm to ocn state mapping file
    atm2ocnSmaptype - atm to ocn state mapping type
    ocn2atmFmapname - ocn to atm flux mapping file
    ocn2atmFmaptype - ocn to atm flux mapping type
    ocn2atmSmapname - ocn to atm state mapping file
    ocn2atmSmaptype - ocn to atm state mapping type
    atm2iceFmapname - atm to ice flux mapping file
```

atm2iceFmaptype - atm to ice flux mapping type
atm2iceSmapname - atm to ice state mapping file
atm2iceSmaptype - atm to ice state mapping type
ice2atmFmapname - ice to atm flux mapping file
ice2atmFmaptype - ice to atm flux mapping type
ice2atmSmapname - ice to atm state mapping file
ice2atmSmaptype - ice to atm state mapping type
atm2lndFmapname - atm to lnd flux mapping file
atm2lndFmaptype - atm to lnd flux mapping type
atm2lndSmapname - atm to lnd state mapping file
atm2lndSmaptype - atm to lnd state mapping type
lnd2atmFmapname - lnd to atm flux mapping file
lnd2atmFmaptype - lnd to atm flux mapping type
lnd2atmSmapname - lnd to atm state mapping file
lnd2atmSmaptype - lnd to atm state mapping type
rof2ocnFmapname - rof to ocn flux mapping file
rof2ocnFmaptype - rof to ocn flux mapping type

cpl_modelio.nml Input File

The cpl_modelio.nml file sets the filename for the primary standard output file. Existance is largely historical. The cpl.buildnml.csh script actually generates the _modelio.nml files for all components.

modelio namelist

`diri (char)`

unused

`diro (char)`

unused

`logfile (char)`

the name of the logfile for the component. default="", required but set automatically by ccsm scripts

drv_in Input File

The drv_in input file contains several different namelist groups associated with general options, time manager options, pe layout, timing output, and parallel IO settings. The different groups are seq_infodata_inparm, seq_timemgr_inparm, ccsm_pes, prof_inparm, and pio_inparm.

seq_infodata_inparm namelist

These namelist are associated with some general driver/coupler options.

`aoflux_grid (char)`

untested

`aqua_planet (log)`

turns on aqua planet mode. this mode is only available in limited configurations. default=false.

`atm_adiabatic (log)`

turns on atm model adiabatic mode. this mode is only available in limited configurations. default=false.

`atm_ideal_phys (log)`

turns on atm ideal physics mode. this mode is only available in limited configurations. default=false.

`bfbflag (log)`

turns on bfb option in coupler which produce bfb results in the coupler on different processor counts. set by BFBFLAG in env_run.xml, default=false.

`brnch_retain_casename (log)`

information about whether the base casename can be reused for the branch casename.

`budget_ann (int)`

sets the diagnotics level of the annual budgets. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_ANNUAL in env_run.xml. default=1

`budget_daily (int)`

sets the diagnotics level of the daily budgets. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_DAILY in env_run.xml. default=0

`budget_inst (int)`

sets the diagnotics level of the instantaneous budgets. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_INST in env_run.xml. default=0

`budget_ltann (int)`

sets the diagnotics level of the longterm budgets written at the end of the year. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_LONGTERM_EOY in env_run.xml. default=1

`budget_ltend (int)`

sets the diagnotics level of the longterm budgets written at the end of each run. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_LONGTERM_STOP in env_run.xml. default=0

`budget_month (int)`

sets the diagnotics level of the monthy budgets. [0,1,2,3], 0=none, 1=+net summary budgets, 2=+detailed lnd/ocn/ice component budgets, 3=+detailed atm budgets, set by BUDGET_MONTHLY in env_run.xml. default=1

`case_desc (char)`

case description. set by CASESTR in env_run.xml.

`case_name (char)`

case name. set by CASE in env_case.xml.

`cpl_cdf64 (log)`

flag for 64 bit netcdf files for the cpl component, may or may not be used by the cpl component depending on cpl implementation. default=false

`do_budgets (log)`

turns on budget calculation and budget output, set by BUDGETS in env_run.xml. default=false

`drv_threading (log)`

turn on run time control of threading per pe per component by the driver. set by DRV_THREADING in env_run.xml. default=false

`eps_aarea (r8)`

variable associated with error checking the atm/lnd areas, absolute differences, set by EPS_AAREA in env_run.xml. default=9.0e-07

`eps_agrid (r8)`

variable associated with error checking the atm/lnd longitudes and latitudes, absolute differences, set by EPS_AGRID in env_run.xml. default=1.0e-12

`eps_amask (r8)`

variable associated with error checking the atm/lnd mask, absolute differences, set by EPS_AMASK in env_run.xml. default=1.0e-13

`eps_frac (r8)`

variable associated with error checking the domain fractions, absolute differences. set by EPS_FRAC in env_run.xml. default=1.0e-02

`eps_oarea (r8)`

variable associated with error checking the ocn/ice areas, absolute differences, set by EPS_OAREA in env_run.xml. default=1.0e-01

`eps_ogrid (r8)`

variable associated with error checking the ocn/ice longitudes and latitudes, absolute differences, set by EPS_OGRID in env_run.xml. default=1.0e-02

`eps_omask (r8)`

variable associated with error checking the ocn/ice mask, absolute differences, set by EPS_OMASK in env_run.xml. default=1.0e-06

`flux_albav (log)`

turns on albedo averaging. set by CPL_ALBAV in env_run.xml, default=false

`flux_epbal (char)`

turns on fresh water balance in coupler, set by CPL_EPBAL in env_run.xml, default=false

`histaux_a2x (log)`

turns on coupler history stream for instantaneous atm to coupler fields. default=false `histaux_a2x24hr (log)`

turns on coupler history stream for daily average atm to coupler fields. default=false

`histaux_a2x3hr (log)`

turns on coupler history stream for 3-hour average atm to coupler fields. default=false

`histaux_a2x3hrp (log)`

turns on coupler history stream for 3-hour average atm to coupler precip fields. default=false

`histaux_l2x (log)`

turns on coupler history stream for instantaneous land to coupler fields. default=false

`histaux_r2x (log)`

turns on coupler history stream for instantaneous runoff to coupler fields. default=false

`hostname (char)`

hostname information, set by MACH in env_case.xml

`info_debug (int)`

debug level in driver/coupler, set by INFO_DBUG in env_run.xml, default=1.

`logFilePostFix (char)`

untested

`model_version (char)`

model version documentation, set by CCSM_REPOTAG in env_run.xml

`npfix (log)`

turns on special north pole fix for u and v vector mapping between atm and ocean grids in the coupler. set by NPFIX in env_run.xml default=true

`ocean_tight_coupling (log)`

turns on ocean tight coupling flag which forces the ocean to coupling at the same time as the other surface models. by default, there is a lag of one coupler timestep in addition to any difference in coupling frequency to accomodate the ability to run the ocean concurrently with the rest of the system for performance reasons. set by OCN_TIGHT_COUPLING in env_run.xml. default=false

`orb_eccen (r8)`

eccentricity of orbit, either the orb_iyear_AD must be set or the other three orb parameter must be set. default=unset

`orb_iyear_AD (int)`

year of orbit, either the orb_iyear_AD must be set or the other three orb parameter must be set. default=unset

`orb_mvelp (r8)`

location of vernal equinox in longitude degrees, either the orb_iyear_AD must be set or the other three orb parameter must be set. default=unset `orb_obliq (r8)`

obliquity of orbit in degrees, either the orb_iyear_AD must be set or the other three orb parameter must be set. default=unset

`outPathRoot (char)`

untested

`perpetual (log)`

flag to turn on perpetual mode, default=false (untested?)

`perpetual_ymd (int)`

ymd date of perpetual mode, default=unset (untested?)

`restart_file (char)`

driver restart file. if set, this overwrites anything in the restart_pfile file. default=unset and uses filename in restart_pfile.

`restart_pfile (char)`

restart pointer filename. default="rpointer.drv"

`samegrid_al (log)`

tells the coupler to treat the atm and land grids as "the same". automatically set by comparison of ATM_GRID and LND_GRID set in env_case.xml. default=true

`samegrid_ao (log)`

tells the coupler to treat the atm and ocean grids as "the same". automatically set by comparison of ATM_GRID and OCN_GRID set in env_case.xml. default=true

`samegrid_ro (log)`

tells the coupler to treat the runoff and ocean grids as "the same". automatically set by comparison of LND_GRID and OCN_GRID set in env_case.xml. default=false

`shr_map_dopole (log)`

turns on special polar mapping feature in shr_map_mod. set by SHR_MAP_DOPOLE in env_run.xml. default=true

`scmlon (r8)`

grid point longitude associated with single column mode. set by PTS_LON in env_run.xml.

`scmlat (r8)`

grid point latitude associated with single column mode. set by PTS_LAT in env_run.xml.

`single_column (log)`

turns on single column mode. set by PTS_MODE in env_case.xml, default=false

`start_type (char)`

mode to start the run up, [startup,branch,continue], set by RUN_TYPE in env_conf.xml

`timing_dir (char)`

location of timing output.

`tchkpt_dir (char)`

location of timing checkpoint output.

`username (char)`

username documentation, set by CCSMUSER in env_run.xml

seq_timemgr_inparm namelist

This namelist is associated with time manager options.

`atm_cpl_dt (int)`

atm coupling timestep in seconds. set via ATM_NCPL in env_run.xml. ATM_NCPL is the number of times the atm is coupled per day.

`atm_cpl_offset (int)`

controls phasing of the coupling. offset in seconds. default=0.

`calendar (char)`

calendar in use. [NO_LEAP, GREOGORIAN]. default="NO_LEAP". set by CALENDAR in env_run.xml

`curr_tod (int)`

untested

`curr_ymd (int)`

untested

`end_restart (log)`

forces a restart write at the end of the run in addition to any setting associated with rest_option. default=true. this setting will be set to false if restart_option is none or never.

`glc_cpl_dt (int)`

glc coupling timestep in seconds. set via GLC_NCPL in env_run.xml. GLC_NCPL is the number of times the glc is coupled per day.

`glc_cpl_offset (int)`

controls phasing of the coupling. offset in seconds. default=0.

`history_n (int)`

number associated with history_option. default=-1. set by HIST_N in env_run.xml.

`history_option (char)`

coupler history snapshot option. see stop_option above. default="never". set by HIST_OPTION in env_run.xml.

`history_ymd (int)`

date associated with history_option date. yyyymmdd format. default=-1. set by HIST_DATE in env_run.xml.

`histavg_n (int)`

number associated with histavg_option. default=-1. set by AVGHIST_N in env_run.xml.

`histavg_option (char)`

coupler time average history option. see stop_option above. defaults="never". set by AVGHIST_OPTION in env_run.xml.

`histavg_ymd (int)`

date associated with histavg_option date. yyyymmdd format. default=-1. set by AVGHIST_DATE in env_run.xml.

`ice_cpl_dt (int)`

ice coupling timestep in seconds. set via ICE_NCPL in env_run.xml. ICE_NCPL is the number of times the ice is coupled per day.

`ice_cpl_offset (int)`

controls phasing of the coupling. offset in seconds. default=0.

`lnd_cpl_dt (int)`

lnd coupling timestep in seconds. set via LND_NCPL in env_run.xml. LND_NCPL is the number of times the lnd is coupled per day.

`lnd_cpl_offset (int)`

controls phasing of the coupling. offset in seconds. default=0.

`ocn_cpl_dt (int)`

ocn coupling timestep in seconds. set via OCN_NCPL in env_run.xml. OCN_NCPL is the number of times the ocn is coupled per day.

`ocn_cpl_offset (int)`

controls phasing of the coupling. offset in seconds. default=0.

`ref_tod (int)`

reference start time of day in seconds. overwritten if restart file is read. if unset, will use start_tod. default=0

`ref_ymd (int)`

reference start date. overwritten if restart file is read. format yyyymmdd. if unset, will use start_ymd. default=0

`restart_n (int)`

number associated with restart_option. default=-1. set by REST_N in env_run.xml

`restart_option (char)`

model restart option. all components write restarts using this setting. see stop_option above. default="yearly". set by REST_OPTION in env_run.xml

`restart_ymd (int)`

date associated with stop_option date. yyyymmdd format. default=-1. set by REST_DATE in env_run.xml `start_tod (int)`

initial start time of day in seconds. overwritten if restart file is read. default=0

`start_ymd (int)`

initial start date. overwritten if restart file is read. format yyyymmdd. default=0, must be set by user on initial runs.

`stop_n (int)`

number associated with stop_option. default=-1. set by STOP_N in env_run.xml

`stop_option (char)`

model stop option used in conjuction with stop_n, stop_ymd, and stop_tod. [none,never,nsteps,nstep,nseconds,nsecond,nminutes,nminute,nhours,nhour,ndays,nday,nmonths,nm
default="". set by STOP_OPTION in env_run.xml

`stop_tod (int)`

time of day in seconds associated with stop_option and stop_ymd. default=0.

`stop_ymd (int)`

date associated with stop_option. model will stop when stop_option and stop_n OR when stop_ymd and stop_tod is reached. yyyymmdd format. default=-1. set by STOP_DATE in env_run.xml

`tprof_n (int)`

number associated with tprof_option. default=-1. set by TPROF_N in env_run.xml.

`tprof_option (char)`

timer profiling option. see stop_option above. default="never". set by TPROF_OPTION in env_run.xml.

`tprof_ymd (int)`

date associated with tprof_option date. yyyymmdd format. default=-1. set by TPROF_DATE in env_run.xml.

ccsm_pes namelist

This namelist is associated with the pe layout.

`atm_ntasks (int)`

the number of mpi tasks assigned to the atm components. set by NTASKS_ATM in env_mach_pes.xml. default=all_pes

`atm_nthreads (int)`

the number of threads per mpi task for the atm component. set by NTHRDS_ATM in env_mach_pes.xml. default=1

`atm_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the atm component. set by PSTRID_ATM in env_mach_pes.xml. default=1 `atm_rootpe (int)`

the global mpi task rank of the root processor assigned to the atm component. set by ROOTPE_ATM in env_mach_pes.xml. default=0

`cpl_ntasks (int)`

the number of mpi tasks assigned to the cpl components. set by NTASKS_CPL in env_mach_pes.xml. default=all_pes

`cpl_nthreads (int)`

the number of threads per mpi task for the cpl component. set by NTHRDS_CPL in env_mach_pes.xml. default=1

`cpl_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the cpl component. set by PSTRID_CPL in env_mach_pes.xml. default=1

`cpl_rootpe (int)`

the global mpi task rank of the root processor assigned to the cpl component. set by ROOTPE_CPL in env_mach_pes.xml. default=0

`glc_ntasks (int)`

the number of mpi tasks assigned to the glc components. set by NTASKS_GLC in env_mach_pes.xml. default=all_pes

`glc_nthreads (int)`

the number of threads per mpi task for the glc component. set by NTHRDS_GLC in env_mach_pes.xml. default=1

`glc_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the glc component. set by PSTRID_GLC in env_mach_pes.xml. default=1

`glc_rootpe (int)`

the global mpi task rank of the root processor assigned to the glc component. set by ROOTPE_GLC in env_mach_pes.xml. default=0

`ice_ntasks (int)`

the number of mpi tasks assigned to the ice components. set by NTASKS_ICE in env_mach_pes.xml. default=all_pes

`ice_nthreads (int)`

the number of threads per mpi task for the ice component. set by NTHRDS_ICE in env_mach_pes.xml. default=1

`ice_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the ice component. set by PSTRID_ICE in env_mach_pes.xml. default=1

`ice_rootpe (int)`

the global mpi task rank of the root processor assigned to the ice component. set by ROOTPE_ICE in env_mach_pes.xml. default=0 `lnd_ntasks (int)`

the number of mpi tasks assigned to the lnd components. set by NTASKS_LND in env_mach_pes.xml. default=all_pes

`lnd_nthreads (int)`

the number of threads per mpi task for the lnd component. set by NTHRDS_LND in env_mach_pes.xml. default=1

`lnd_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the lnd component. set by PSTRID_LND in env_mach_pes.xml. default=1

`lnd_rootpe (int)`

the global mpi task rank of the root processor assigned to the lnd component. set by ROOTPE_LND in env_mach_pes.xml. default=0

`ocn_ntasks (int)`

the number of mpi tasks assigned to the ocn components. set by NTASKS_OCN in env_mach_pes.xml. default=all_pes

`ocn_nthreads (int)`

the number of threads per mpi task for the ocn component. set by NTHRDS_OCN in env_mach_pes.xml. default=1

`ocn_pestride (int)`

the mpi global processors stride associated with the mpi tasks for the ocn component. set by PSTRID_OCN in env_mach_pes.xml. default=1

`ocn_rootpe (int)`

the global mpi task rank of the root processor assigned to the ocn component. set by ROOTPE_OCN in env_mach_pes.xml. default=0

prof_inparm namelist

This namelist is associated with timing profile and is namelist for the perf_mod timing utility.

`profile_barrier (log)`

`profile_depth_limit (int)`

`profile_detail_limit (int)`

`profile_disable (log)`

`profile_global_stats (log)`

`profile_outpe_num (int)`

`profile_outpe_stride (int)`

`profile_papi_enable (log)`

`profile_single_file (log)`

`profile_timer (int)`

pio_inparm namelist

This namelist is associated with pio setting for driver/coupler IO.

`cpl_io_numtasks (int)`

the number of mpi tasks to assign to pio IO. default=coupler_pes/4

`cpl_io_root (int)`

the coupler mpi task associated with the root task of the pio processor group. default=0

`cpl_io_stride (int)`

the mpi stride of the pio tasks. default=4

`cpl_io_typename (char)`

the type of IO implementation to use. [netcdf, pnetcdf]. default=netcdf

seq_maps.rc Input File

The seq_maps.rc file specifies the mapping files for the configuration and is generated by the cpl.buildnml.csh file. It is NOT a Fortran namelist file but the format should be relatively clear from the default settings.

seq_maps input variables

`atm2ocnFmapname (char)`

The atm to ocn mapping file for flux fields. Set by MAP_A2OF_FILE in env_conf.xml idmap is a special string used here to specify that just a copy is required.

`atm2ocnFmaptype (char)`

The type of mapping desired, either "source" or "destination" mapping. X is associated with rearrangement of the source grid to the destination grid and then local mapping. Y is associated with mapping on the source grid and then rearrangement and sum to the destination grid.

`atm2ocnSmapname (char)`

see atm2ocnFmapname above but for atm to ocn scalar fields using env variable MAP_A2OS_FILE

`atm2ocnSmaptype (char)`

see atm2ocnFmaptype above

`ocn2atmFmapname (char)`

see atm2ocnFmapname above but for ocn to atm flux fields using env variable MAP_O2AF_FILE

`ocn2atmFmaptype (char)`

see atm2ocnFmaptype above

`ocn2atmSmapname (char)`

see atm2ocnFmapname above but for ocn to atm scalar fields using env variable MAP_O2AS_FILE

`ocn2atmSmaptype (char)`

see atm2ocnFmaptype above

`atm2iceFmapname (char)`

see atm2ocnFmapname above but for atm to ice flux fields using env variable MAP_A2IF_FILE

`atm2iceFmaptype (char)`

see atm2ocnFmaptype above

`atm2iceSmapname (char)`

see atm2ocnFmapname above but for atm to ice scalar fields using env variable MAP_A2IS_FILE

`atm2iceSmaptype (char)`

see atm2ocnFmaptype above

`ice2atmFmapname (char)`

see atm2ocnFmapname above but for ice to atm flux fields using env variable MAP_I2AF_FILE

`ice2atmFmaptype (char)`

see atm2ocnFmaptype above

`ice2atmSmapname (char)`

see atm2ocnFmapname above but for ice to atm scalar fields using env variable MAP_I2AS_FILE

`ice2atmSmaptype (char)`

see atm2ocnFmaptype above

`atm2lndFmapname (char)`

see atm2ocnFmapname above but for atm to lnd flux fields using env variable MAP_A2LF_FILE `atm2lndFmaptype (char)`

see atm2ocnFmaptype above

`atm2lndSmapname (char)`

see atm2ocnFmapname above but for atm to lnd scalar fields using env variable MAP_A2LS_FILE

`atm2lndSmaptype (char)`

see atm2ocnFmaptype above

`lnd2atmFmapname (char)`

see atm2ocnFmapname above but for lnd to atm flux fields using env variable MAP_L2AF_FILE

`lnd2atmFmaptype (char)`

see atm2ocnFmaptype above

`lnd2atmSmapname (char)`

see atm2ocnFmapname above but for lnd to atm scalar fields using env variable MAP_L2AS_FILE

`lnd2atmSmaptype (char)`

see atm2ocnFmaptype above

`rof2ocnFmapname (char)`

see atm2ocnFmapname above but for runoff to ocean fields using env variable MAP_R2O_FILE_R05, MAP_R2O_FILE_R019, or MAP_R2O_FILE_RX1 depending on the configuration.

`rof2ocnFmaptype (char)`

see atm2ocnFmaptype above