# The CCSM Coupler
Version 5.0.1

Combined

**User's Guide,**
**Source Code Reference,**
and **Scientific Description**

Brian G. Kauffman
William G. Large

August 2002

**Community Climate System Model**
National Center for Atmospheric Research, Boulder, CO
http://www.ccsm.ucar.edu/models

CVS tag `$Name: cpl5_0_1 $` Build date: October 10, 2002

# Contents

# 1 Introduction

## 1.1 Purpose of this document

```
CCSM components typically are accompanied by three types of documents:

1) A User's Guide -- information about how do compile and execute the code
2) A Code Reference -- information about source code and how to do source code
   modifications
3) A Scientific Description -- information about the physical and numerical
   equations, algorithms, and constants that the source code implements.

This document combines the content of all three documents for the CCSM
Coupler component.  Similar documents exist for the CCSM's atmosphere, land,
ocean, and sea-ice components.  Also, there is a CCSM User's Guide which
explain how to compile and execute the CCSM system as a whole.
```

## 1.2 What is a "coupler" ?

```
The CCSM coupled model is a framework that divides the complete climate system
into component models connected by a coupler.  In this design the Coupler
is a hub that connects four component models -- atmosphere, land, ocean, and
sea-ice (see Figure 1).  Each component model is connected to the Coupler, and
each exchanges data with the Coupler only. From a software engineering point of
view, the CCSM is not a particular climate model, but a framework for building
and testing various climate models for various applications. In this sense,
more than any particular component model, the Coupler defines the high-level
design of CCSM software.
```

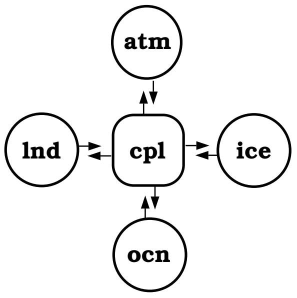

Figure 1: The CCSM Coupled Model Framework in its most basic form.

The Coupler code has several key functions within the CCSM framework:

o It allows the CCSM to be broken down into separate components, atmosphere, sea-ice, land, and ocean, that are "plugged into" the Coupler. Each component model is a separate code that is free to choose its own spatial resolution and time step. Individual components can be created, modified, or replaced without necessitating code changes in other components. CCSM components run as separate executables communicating via message passing (MPI).

o It controls the execution and time evolution of the complete CCSM by synchronizing and controlling the flow of data between the various components.

o It communicates interfacial fluxes between the various component models while insuring the conservation of fluxed quantities. For certain flux fields, it also computes interfacial fluxes based on state variables. In general, the Coupler allows any given flux field to be computed once in one component, this flux field is then routed through the Coupler so that other involved components can use this flux field as boundary forcing data.

## 1.3 Major considerations in Coupler functionality

The design of the Coupler and the CCSM modeling framework were motivated by a variety of scientific and software design issues. Following are some of the major design and functionality considerations which addressed the deficiencies of standard coupling strategies at the time of Coupler's inception (circa 1991), and which have continued to be important considerations.

(a) System decomposition and component model code independence.

The CCSM coupling strategy and the Coupler component (see Figure 1) allows an intuitive decomposition of the large and complex climate system model. To a large extent, the separate component models, atmosphere, sea-ice, land, and ocean, can be designed, developed, and maintained independently, and later "plugged into" the Coupler to create a complete climate system model. At the highest design level, this creates a highly desirable design trait of creating natural modules (or "objects") with maximum internal cohesion and minimal external coupling.

(b) Control of the execution and time evolution of the system.

The Coupler (a natural choice), rather than one of the component models (any of which would be an arbitrary and asymmetrical choice), is responsible for controlling the execution and time evolution of the complete system.

(c) The appropriate boundary condition for all component models are the fluxes across the surface interfaces.

In terms of the climate simulation, the Coupler (originally called the "Flux Coupler") couples component models by providing flux boundary conditions to all component models. State variables flow through the Coupler only if necessary for a flux calculation in the Coupler or a component model.

(d) Conservation of fluxed quantities.

The Coupler can conserve all fluxed quantities that pass through it. Since all surface fluxes pass through the Coupler, this framework assures that all surface fluxes in the system are conserved. The Coupler also can and does monitor flux conservation by doing spatial and temporal averages of all fluxed quantities.

(e) A flux field should should be computed only once, in one component, and then the field should be given to other components, as necessary.

For example, generally the resolution of the ocean component is finer than that of the atmospheric component. Heat is not conserved if the atmosphere component computes long wave radiation using sea surface temperature (SST) averaged onto the atmosphere's grid, while the ocean component uses individual grid point SSTs. This is because the flux calculation is proportional to SST**4 (is non-linear) and the two calculations will yield two different results . Instead of trying to compute the same flux twice, on two different grids in two different components, the flux should be computed once and then be mapped, in a conservative manner, to other component grids.

(f) Fluxes can be computed in the most desirable place.

For example, when an atmospheric grid box covers both land and ocean, a problem arises if wind stress is first computed on the atmospheric grid and then interpolated to the ocean grid. If land roughness is significantly larger than ocean roughness (as is typical), and the atmosphere uses an average underlying roughness to compute the windstress, and then windstress is interpolated to coastal land and ocean cells, windstress will be considerably lower (for land) or higher (for ocean) than it should be. This approach can lead to unrealistic coastal ocean circulations. In this case it is more desirable to compute the surface stresses on the surface grids and subsequently merge and map them to the atmospheric grid. Thus this calculation should take place in either the Coupler, land, ice, or ocean models.

In the case of computing precipitation, because this calculation requires full 3D atmospheric fields, and because only 2D surface fields are exchanged through the Coupler, this calculation should take place in the atmosphere component.

(g) Allowing the coupling of component models with different spatial grids.

In general it is desirable to allow the different component models to exist on different spatial grids. It is also desirable that a component model need not be aware of, or concerned with, the spatial grids that other component models are using. The Coupler handles all mapping between disparate model grids, allows component models to remain unaware and unconcerned with the spatial grids of other CCSM components.

Due to scientific considerations, the current version of CCSM requires that the atmosphere and land components be on the same spatial grid. Because it is not a requirement to do so, the Coupler does not have the functionality to allow the atmosphere and land to be on different grids. Similarly, the current version of CCSM requires that ocean and sea-ice components be on the same grid and the Coupler does not have the functionality to allow them to be on different grids.

Except as noted above, no component needs to know what spatial grid other components are using. The Coupler is responsible for all mapping between the various spatial grids of the various components models. The component models themselves have no knowledge of what other grids are involved in the coupled system and they can remain unconcerned with any issues regarding mapping between grids.

(h) Allowing the coupling of models with different time steps.

While there are some restrictions on what internal time step models are allowed to use (for example, there must be an integer number of time steps per day), the Coupler offers component models considerable freedom in choosing their internal time step. Frequently all four CCSM components operate with four different internal time steps.

## 1.4 History

The origins of the Coupler are in the Oceanography Section (OS) of the Climate and Global Dynamics Division (CGD) of the National Center for Atmospheric Research (NCAR). While coupling the Community Climate Model (CCM2, the atmosphere model) with a regional Pacific Basin Model for ESNO/El-Nino studies (Gent, Tribbia, Kauffman, & Lee, 1990), it became clear that it was extremely awkward to couple the models by implementing the ocean model as a subroutine within an atmosphere model. An idea emerged that a more desirable software architecture would be to have both the ocean and atmosphere models as subroutines to a higher-level driver/coupler program. This "coupler" component would reconcile the differing component model spatial grids (handle the mapping of data fields), reconcile different time step intervals, and control when the coupled system would start and stop. Thus the two component models could be very independent and largely unaware of the software details of the other components. The modularity of this arrangement also suggested a framework in which, for example, it would be relatively easy to exchange one ocean component for another.

Other OS scientists joined in (McWilliams, Large, Bryan, 1991), and together worked out a scientific design philosophy for the Coupler and the coupled system, for example, that the appropriate boundary conditions for component models were the fluxes across the surface interfaces, that fluxes should be conserved, and that ad hoc flux corrections were undesirable. Frank Bryan suggested the models be separate executables communicating by message passing rather than subroutines in a single executable, this resulted in an MPMD implementation that proved extremely successful.

Climate Modeling Section (CMS) scientists joined the effort (Boville, et.al., 1992), working on the atmosphere component (CCM) so that it produced a quality simulation when bottom boundary conditions were fluxes instead of fixed BC's. The CMS also had experience in managing community modeling efforts -- the CMS managed the development of the CCM, a very successful community climate model. With the Coupler and its associated framework, and with a critical mass of engaged ocean and atmosphere model scientists, it now seemed plausible to start an institutional coupled climate model project.

A proposal was made to NSF for a new, NCAR-wide, Climate System Model (CSM) Project (1993), with the long-term goal of building, maintaining, and continually improving a comprehensive model of the climate system. The Coupler proto-type code and design philosophy formed the basis for CSM's software framework. The CSM project was funded and formally began in 1994.

The name, "cpl5" or "Coupler, version 5", alludes to five versions of the Coupler, they are:

cpl1 (Kauffman, 1990): a rough proto-type code, never released.
  Coupler is a driver program that handles all high-level control (eg. start/stop/advance of component models) and all mapping between grids. The complete system is a single-executable with two swappable component model subroutines: ocean/ice and atmosphere/land. Written in Fortran 77.

cpl2 (NCAR/CGD, Oceanography Section, 1992): a proto-type code, never released.
  New scientific design philosophy, does flux calculations, insures conservation of fluxed quantities. Complete system uses component models

that are separate executables. Uses PVM for message passing.

cpl3 (NCAR/CGD, 1996): released with CSM 1.0
Has three component models: atmosphere/land, ocean, and ice. First fully functional version. Uses MCL for message passing. Uses netCDF.

cpl4 (NCAR/CGD, 1998): released with CSM 1.2
Has four component models: atmosphere, land, ocean, and ice. Uses MPI for message passing.

cpl5 (NCAR/CGD, 2002): released with CSM 2.0
Written in Fortran 90, uses SCRIP to generate mapping data, handles shifted pole grids.

# 2 Running the Coupler

The Coupler cannot run by itself, it can only execute in the context of running the complete CCSM system -- a framework that requires atmosphere, ice, land, and ocean components. The scripts that build and run the CCSM system are described in detail in the CCSM 2.0.1 User's Guide, a holistic guide to running the complete system. This Guide includes a line-by-line explanation of the master run script and the Coupler's "setup script". A brief description of CCSM run scripts is below. See the CCSM User's Guide for complete information.

## 2.1 Master Run Script and Coupler Setup Script

Two levels of c-shell scripts are used to build and run the CCSM system, which of course, includes the Coupler. A master "run script" coordinates the building and running the complete system while the component model "setup scripts" are responsible for configuring each individual CCSM component (including the Coupler). The CCSM execution is controlled by the master script, referred to as "the run script". The run script has the following tasks:

a. Set batch system options
b. Define common build and run environment variables
c. Select multi-processing and resolution specs
d. Run the setup script for each component
e. Run the CCSM Integration.
f. Archive/harvest/resubmit when this run is finished

The common build and run environment variable defined in the run script are automatically propagated to each of the component model setup scripts. These variables define such things as the machine architecture, the number of CPUs to run on, common experiment and file naming conventions.

Once the master run script has defined the common environment, each of the component models (cpl, atm, ice, lnd, and ocn) are configured using individual component setup scripts (e.g. cpl.setup.csh) which:

```
   a. Parse the environment variables sent from the master run script. These
      are, in effect, input variables that might be required by a setup script
      or otherwise might alter the behavior of the setup script.
   b. Position or create any input data files, as necessary, including input
      namelist data files and mapping data files.
   c. Build the component executable.
```

Finally, when all of the component model setup scripts have successfully completed, the run script executes all CCSM components simultaneously. The CCSM component models run simultaneously as a Multi-Program/Multi-Data (MPMD) message passing system, using MPI to exchange data.

# 3 Input Namelist Parameters

To some extent, the behavior of the Coupler can be specified or modified at run time. This is done almost exclusively through the use of an F90 namelist. This section contains a list and description of available namelist input parameters. The Coupler reads an input namelist, called inparm, from stdin, at runtime.

## 3.1 List of Parameters

Coupler input variables follow a naming convention in which the first few letters of the variable identify a basic functionality group:

```
   * case_xxx  selects options with respect to specifying a case name and
               description
   * rest_xxx  selects options with respect to the creation or retrieval of
               restart files and to specify related initial conditions.
   * stop_xxx  selects options with respect to when a simulation will stop
   * hist_xxx  selects options with respect to when and what type of
               history data is created
   * diag_xxx  selects options with respect to run-time diagnostics of the
               physical simulation,
   * flx_xxx   selects options with respect to flux calculation specifics
   * orb_xxx   selects options with respect to solar orbit calculations
   * mss_xxx   selects options with respect to data file archival (in the past
               this meant using NCAR's Mass Storage System, the "MSS")
   * info_xxx  selects options with respect to monitoring the progress or
               computational performance of the model

The following list of input namelist parameters includes this information:

   * Type: the variable's data type.
     Notes:
     o some character strings have length = CLONG.  "CLONG" is a single global
       constant used throughout the Coupler to define a "long" character string.
       The CLONG is hardcoded to be 256.
     o "integer boolean" means an integer such that a value of zero is
       interpreted as "false", and any non-zero value is interpreted as "true".
   * Default: the default value of the variable. While reasonable default values
     are selected when possible, users might want to alter these values
     according to their application.
   * Required: tells if and when the variable is required input.  Very few
     input parameters are required. In a few cases, changing one default value
     will cause other input parameters to become required.
   * Description: a brief description of the purpose and effect of the
     parameter.
   * Example: a reference to an example namelist which uses the variable
     (examples follow this list).

case_name
     Type: character(len=CLONG)
     Default: "null"
     Required: no, but highly recommended
     Description: This is the case name text string which is used to create
     output file names and is also included in output files to help identify the
     model run.  Generally this should be a short string (eg. 8 chars) and
     contain only those characters that are valid in unix file names.
     See Example: 1, 2, 3, 4

case_desc
     Type: character(len=CLONG)
     Default: "null"
     Required: no, but highly recommended
     Description: This is a short text string (typically less than 80 chars)
     which is included in output files to help identify the model run.
     See Example: 1, 2, 3, 4

rest_type
```

```
     Type: character(len=32)
     Default: "startup"
     Required: no (but default is of limited usefulness)
     Description: This selects the run type.
     Valid choices are: "startup", "hybrid", "continue", or "branch".
     Selecting "branch" makes rest_bfile a required input.
     See Example: 1, 2, 3, 4

rest_pfile
     Type: character(len=CLONG)
     Default: "null"
     Required: yes
     Description: This is the complete path and name of the restart "pointer
     file."  This must include an existing, NFS mounted directory. All run
     types will update this file (and create it, if necessary), but only a
     continuation runs requires that this file exists prior to the start of
     the run.
     See Example: 1, 2, 3, 4

rest_bfile
     Type: character(len=CLONG)
     Default: "null"
     Required: required if rest_type = "branch", ignored otherwise.
     Description: This is the file name of the "branch file" (the IC data file).
     Note that a prefix like "mss:" is used to indicate a file archival device,
     Valid prefix options are:
     * "cp:" or no-prefix indicates a normal unix file copy from an NFS mounted
        file system.
     * "mss:" indicates a file on NCAR's MSS
     * "null:" indicates no archival -- the file name, stripped of any directory
        information, indicates a file in the current working directory.
     See Example: 4

rest_date
     Type: integer
     Default: 00000101 (year 0, month 1, day 1, encoded yyyymmdd)
     Required: no (ignored for "continuation" runs, optional for others).
     Description: This is the restart date.
     * On startup and hybrid runs, rest_date is the initial date of the
       simulation.
     * On branch runs, if rest_date > 0, it will over-ride the date contained
       in the rest_bfile IC restart file. Otherwise the date from the
       rest_bfile data file is used.
     * On continuation runs rest_date is not used (the date contained in the IC
       file is used).
     See Example: 1, 2
```

```
rest_freq
     Type: character(len=32)
     Default: "monthly"
     Required: no
     Description: This is the restart frequency which selects how often restart
     data files are created.  Options are:
     * "yearly"    => restart files on every January 1st
     * "halfyear   => restart files on every January 1st and July 1st
     * "quarterly" => restart files on every January, April, July, & October 1st
     * "monthly"   => restart files on the 1st of every month
     * "ndays"     => restart files every n days
     If rest_freq = "ndays", then namelist parameters rest_n and rest_odate are
     involved (see below).
     See Example: 4

rest_n
     Type: integer
     Default: 10
     Required: no (only used if rest_freq = "ndays")
     Description: If rest_freq = "ndays", restart files will be created
     every n days.  More specifically, whenever mod(d-p;n) = 0, where
     d = the simulation day (1 Jan, year 1 <=> d = 365)
     p = an offset such that mod(d-p;n) = 0 on the date specified by rest_odate
     n = rest_n
     See Example: 4

rest_odate
     Type: integer
     Default: 0
     Required: no
     Description: Selects restart file offset calendar date, used in
     conjunction with rest_n when rest_freq = "ndays." If rest_odate > 0, it
     must be a valid calendar date (encoded: yyyymmdd).  Restart data is then
     created every n days relative to (and including) either
     * the initial date of the run (if rest_odate < 1)
     * the date rest_odate (if rest_odate > 0)

stop_option
     Type: character(len=16)
     Default: "newyear"
     Required: no
     Description: Selects when to stop the simulation.
     * "newdecade" => stop on the January 1st at the start of the next decade
     * "newyear"   => stop on the next January 1st
     * "halfyear   => stop on the next January 1st or July 1st
     * "newmonth"  => stop on the 1st of the next month
     * "nmonths"   => stop on day 1, n months from this month, where n=stop_n
     * "ndays"     => stop after advancing n days, where n=stop_n
     * "date"      => stop when stop_date is reached. This makes stop_date a
                      required input.
     See Example: 1, 2, 3, 4

stop_n
     Type: integer
     Default: 0
     Required: only if stop_option = "ndays" or "nmonths"
     Description: If stop_option = "ndays", stop after advancing stop_n days.
     If stop_option = "nmonths", stop on day 1, n months from this month.
     See Example: 4

stop_date
     Type: integer
     Default: 00010101 (year 1, month 1, day 1)
     Required: no (required if stop_option = "date", ignored otherwise)
     Description: If stop_option = "date", stop on this date.
     See Example: 2

hist_freq
     Type: character(len=16)
     Default: "never"
     Required: no
     Description: selects how often instantaneous history data is created:
     * "yearly"    => data is created on the 1st of each January
     * "quarterly" => data is created on the 1st of January,April,July,October
     * "monthly"   => data is created on the 1st of each month
     * "biweekly"  => data is created on the 1st and 15th of each month
     * "weekly"    => data is created on the 1st, 8th, 15th, and 22nd of each
                      month
     * "daily"     => data is created every day
     * "ndays"     => history data is created every n days relative to a history
                      offset date specified by hist_odate.
     * "nstep"     => history data is created every n time steps relative to and
                      including the first time step of each day.
     Note: history data is never created on a simulation stopping date, but
     history data can be created on a simulation starting date.
     See Example: 3

hist_n
     Type: integer
     Default: 1
     Required: no (only used if hist_freq = "ndays" or "nstep")
     Description: If hist_freq = "ndays" > 0, history files will be created
     every n days. More specifically, whenever mod(d-p;n) = 0, where
     d = the simulation day (1 Jan, year 1 <=> d = 365 )
     n = hist_n
     p = an offset such that mod(d-p;n) = 0 on the date specified by hist_odate
     If hist_freq = "nstep" and hist_n > 0, history files will be created every
     n time steps including the first time step of each day.
     See Example: 3

hist_odate
     Type: integer
     Default: 1
     Required: no (only used if hist_freq = "ndays")
     Description: Selects history file offset calendar date, used in
     conjunction with hist_n when hist_freq = "ndays". If hist_odate > 1, it
     must be a valid calendar date (encoded: yyyymmdd). See hist_freq for usage.
     History data is created every n days relative to (and including) either
     * the initial date of the run, if hist_odate < 1
     * the 1st of the every month , if hist_odate = 1
     * the date hist_odate        , if hist_odate > 1
     See Example: 3

hist_bundle
     Type: character(len=16)
     Default: "null" (no bundling)
     Required: no
     Description: If hist_bundle is "daily", "monthly", "quarterly", or
     "yearly", then instantaneous history data files will accumulate multiple
     time samples until the next day, month, quarter, or year (respectively) is
     encountered, then the data file will be closed and archived, and a new
     history file will be created, and subsequent data will accumulate into this
     new file. A history file's name indicates the date of the first time sample
     in that file.

hist_tavg
     Type: integer
     Default: 1 (true)
     Required: no
     Description: If hist_tavg is true (non-zero), the Coupler will create
     monthly average history data files and component models will be requested
     to do the same.  Time averaged data is not bundled (i.e. there is always
     one set of monthly average fields per file).

diag_freq
     Type: character(len=16)
     Default: "never"
     Required: no
     Description: selects how often instantaneous diagnostic data is created:
     * "never"     => data is never created
     * "yearly"    => data is created on the 1st day of each year
     * "quarterly" => data is created on the 1st day of each quarter
     * "monthly"   => data is created on the 1st of each month
     * "biweekly"  => data is created on the 1st and 15th of each month
     * "weekly"    => data is created on the 1st, 8th, 15th, and 22nd of each
                      month
     * "ndays"     => data is created every n days relative to an offset date
                      specified by diag_odate. See diag_n.
     * "nstep"     => data is created every n time steps, including the first
                      time step of each day. See diag_n.
     Note: if diagnostics are being printed out on a given day, they are always
     printed out when seconds = 0, 21600, 43200, and 64800 (ie. every six
     hours).  Seconds = 0 will always occur, but depending on what model
     communication intervals are used, it may be that seconds never equals
     21600, 43200, or 64800, and hence there may be as few as one printout per
     day.
     See Example: 3

diag_n
     Type: integer
     Default: 10
     Required: no (only used if diag_freq = "ndays" or "nstep")
     Description: If diag_freq = "ndays", diagnostic data will be created
     every n days. More specifically, whenever mod(d-p;n) = 0, where
     d = the simulation day (1 Jan, year 1 <=> d = 365 )
     n = diag_n
     p = an offset such that mod(d-p;n) = 0 on the date specified by diag_odate.
     If diag_freq = "nstep", diagnostic data will be created every n time steps
     including the first time step of each day.

diag_odate
     Type: integer
     Default: 1
     Required: no (only used if diag_freq = "ndays")
     Description: Selects diagnostic data offset calendar date, used in
     conjunction with diag_n when diag_freq = "ndays". If diag_odate > 1, it
     is a valid calendar date (encoded: yyyymmdd). See diag_freq for usage.
     Diagnostic data is created every n days relative to (and including) either
     * the initial date of the run (if hist_odate < 1)
     * the 1st of the month (if hist_odate = 1)
     * the date hist_odate (if hist_odate > 1)

flx_albav
```

```
     Type: integer boolean
     Default: 0 (false)
     Required: no
     Description: The Coupler computes ocean albedos and sometimes modifies
     ice albedos.
     * if flx_albav = true (non-zero), the ocean albedos are computed such that
       they have no zenith angle dependence and ice albedos, which are computed
       by the ice model, are unaltered.
     * if flx_albav = false (zero), the ocean albedos are computed with a zenith
       angle dependence and ice albedos are set to unity on the dark side of the
       earth.
     Typically flx_albav ("daily average albedos") is only turned on when the
     atm component is a climatological data atm model that is sending daily
     average data to the coupler, and thus "daily average albedos" are an
     appropriate complement to the daily average solar fluxes sent by the
     atm component.

flx_epbal
     Type: character(len=16)
     Default: "off"
     Required: no
     Description: Non-default values can be used to conserve globally integrated
     salinity in ocn & ice components that are coupled to a climatological data
     atm model.  The conservation takes place by multiplying precipitation, P,
     and runoff, R, by a single scalar (spatially constant) factor f to balance
     evaporation, E: <E> + f<P> + f<R> = 0, where <x> denotes a globally
     averaged value.  There are three valid values for flx_epbal:
     * "off"  => no factor is applied to P + R
     * "inst" => the Coupler computes a factor f so that <E> + f<P> + f<R> = 0
                 at each time step ("instantaneously").
     * "ocn"  => the ocn component provides the Coupler with a factor f and the
                 Coupler applies this factor to P and R.  Typically this factor
                 is chosen, by the ocn component, so that <E> + f<P> + f<R> = ,
                 but perhaps not instantaneously, maybe as an annual average.
                 This can be used to allow some fluctuation in the global
                 average of salinity on shorter time scales while enforcing a
                 constant global average of salinity on longer time scales.
                 The ocn component is responsible for providing an appropriate
                 factor to the Coupler.
orb_year
     Type: integer
     Default: <none>
     Required: yes
     Description: This is the calendar year which is used to determine the
     solar orbit and resulting solar angles. This is necessary, for example,
     to compute ocn surface albedo, which may be zenith angle dependent.
     Valid values are in the range [-1000000, +1000000].
     A typical value might be 1990.
     See Example: 1, 2, 3, 4

mss_dir
     Type: character(len=CLONG)
     Default: "null"
     Required: yes
     Description: This is a file archival directory name including the
     specification of an archival device.  Restart and history files go into
     this directory.  For continuation runs, the initial condition restart file
     must also be in this directory.  A "prefix" is all the characters up to
     and including the first occurrence of ":".  The prefix of the file
     indicates a storage device.  Valid prefix options are:
     * "cp:"   or no-prefix indicates a normal unix file copy to an NFS mounted
               file system.
     * "mss:"  indicates archival onto NCAR's MSS
     * "null:" indicates no archival of any sort.
     Some of the following mss_xxx options are only meaningful when the
     archival device is NCAR's MSS.  With code modification, more archival
     devices could be implemented (eg. "hpss:").
     See Example: 1, 2, 3, 4

mss_opts
     Type: character(len=64)
     Default: "nowait, nomail"
     Required: no
     Description: Options used when data files are sent to NCAR's MSS.
     This default request uses asynchronous data file transfer.
     See the msrcp man pages at NCAR for more details.
     See Example: 2

mss_pass
     Type: character(len=16)
     Default: " "  (no password)
     Required: no
     Description: Mswrite write password string for files sent to the MSS.
     Read passwords are not an option with the Coupler.
     See the mswrite man pages at NCAR for more details.
     See Example: 2

mss_rtpd
     Type: integer
     Default: 365
     Required: no
     Description: Retention period, in days, for data files sent to the MSS.
     See the mswrite man pages at NCAR for more details.
     See Example: 2

mss_rmlf
     Type: integer boolean
     Default: 0 (false)
     Required: no
     Description: "remove local file" after archival.  True means local files
     will be removed after they are archived.  Note that the most recently
     created file is never removed, but older files are.
     WARNING: successful archival is not verified prior to removal.
     WARNING: using a "null:" prefix for mss_dir and mss_rmlf = TRUE will
     result in files being deleted but not archived.
     See Example: 2

info_dbug
     Type: integer
     Default: 1
     Required: no
     Description: Debugging information level: 0, 1, 2, or 3.
     * 0 => do not write any extra debugging information to stdout
     * 1 => write a small amount of extra debugging information to stdout
     * 2 => write a medium amount of extra debugging information to stdout
     * 3 => write a large amount of extra debugging information to stdout
     See Example: 3

info_time
     Type: integer
     Default: 0 (false)
     Required: no (normally not recommended)
     Description: If true (non-zero), write message passing timing info to
     stdout.

nx_a,ny_a
nx_i,ny_i
nx_l,ny_l
nx_o,ny_o
     Type: integer
     Default: <none>
     Required: yes
     Description: the "i" and "j" 2d-resolution of the atmosphere, ice, land,
     and ocean components, respectively.  This is with respect to the data
     fields that are sent and received from the Coupler.
     See Example: 1, 2, 3, 4
```

## 3.2 Example Input Namelists

Example 1: a "startup" run

```
    $inparm
    case_name   = "test.01"
    case_desc   = "testing a startup run "
    rest_type   = "startup"
    rest_pfile  = "$HOME:/cpl.test.01.rpointer"
    rest_date   = 19800101
    stop_option = "date"
    stop_date   = 19800701
    orb_year    = 1990
    mss_dir     = "null:/whatever"
    nx_a = 128, ny_a = 64,  nx_i = 192, ny_i = 94
    nx_l = 128, ny_l = 64,  nx_o = 192, ny_o = 94
    /
```

Here the inputs specify a "startup" run starting on 1980 Jan 1st and stopping on 1980 Jul 1st. No initial condition data is needed. A restart pointer file, rest_pfile, will be created in the user's home directory. The restart pointer file will contain the name of the most recently created restart file.
Restart and history files will not be archived in any way -- they will remain in the current working directory.
History data files and restart files will be created at the default frequencies.
Solar orbit calculations will be based on the year 1990.
The "null:" prefix on mss_dir specifies that there will be no archival of history and restart files -- the files will simply remain in the current working directory. Presumably the output files will be archived later in some post-processing phase.

Example 2: a "hybrid" run

```
    $inparm
    case_name   = "test.01"
    case_desc   = "testing a hybrid run"
    rest_type   = "hybrid"
    rest_pfile  = "$HOME/cpl.test.01.rpointer"
    rest_date   = 19800101
    stop_option = "date"
    stop_date   = 19800701
    orb_year    = 1990
    mss_dir     = "mss:/DOE/csm/test.01/cpl/ "
    mss_rtpd    = 730
    mss_pass    = "rosebud"
    mss_opts    = "nowait"
    mss_rmlf    = 1
    nx_a = 128, ny_a = 64,  nx_i = 192, ny_i = 94
    nx_l = 128, ny_l = 64,  nx_o = 192, ny_o = 94
    /
```

As far as the Coupler is concerned, a "hybrid" run is the same as an "initial" run. The distinction is for the benefit of other component models. The stop options will cause the simulation to stop on a particular date, 1981 Aug 1st. The "mss:" prefix on mss_dir selects NCAR's MSS as the archival device for output files. The mass store file retention period (2 years) and write password ("rosebud") have been selected here. Files will be deleted after being archived to the MSS (except that the most recently created file is always left in the current working directory).
The resolution of the atm and lnd grids is 128x64.
The resolution of the ice and ocn grids is 192x94.

Example 3: a "continuation" run

```
    $inparm
    case_name   = "test.01"
    case_desc   = "testing a continuation run "
    rest_type   = "continue"
    rest_pfile  = "$HOME/cpl.test.01.rpointer"
    stop_option = "newyear"
    hist_freq   = "ndays"
    hist_n      = 10
    hist_odate  = 1
    diag_freq   = "monthly"
    orb_year    = 1990
    mss_dir     = "cp:~/test.01/cpl/ "
    info_dbug   = 0
    nx_a = 128, ny_a = 64,  nx_i = 192, ny_i = 94
    nx_l = 128, ny_l = 64,  nx_o = 192, ny_o = 94
    /
```

Here the inputs specify a continuation run. Assuming this run continues from where example 1 finished, this run will start on 1980 Jul 1st and stop on 1981 Jan 1st. Exactly where this run continues from is specified by the restart file, and the restart file which will be used is specified by the restart pointer file. Restart files will be created according to the default setting of rest_freq. History data will be created every 10 days relative to the 1st day of each month, i.e. on the 1st, 11th, 21st, and 31st of every month. Setting diag_freq = "monthly" signals the coupler to calculate some global diagnostic information on the 1st of each month and write it to stdout. New restart and history files will be archived by copying them into a subdirectory of the user's home directory (~/test.01/cpl/). Setting info_dbug = 0 will minimize the amount of debugging information written to stdout.

Example 4: a "branch" run

```
    $inparm
    case_name   = "test.02"
    case_desc   = "testing a branch run "
    rest_type   = "branch"
    rest_pfile  = "$HOME/cpl.test.02.rpointer"
    rest_bfile  = "mss:/DOE/csm/test.01/cpl/test.01.cpl5.r.1981-01-01-00000 "
    rest_freq   = "ndays"
    rest_n      = 10
    stop_option = "ndays"
    stop_n      = 31
    orb_year    = 1990
    mss_dir     = "~/test.01/cpl/ "
    nx_a = 128, ny_a = 64,  nx_i = 192, ny_i = 94
    nx_l = 128, ny_l = 64,  nx_o = 192, ny_o = 94
    /
```

Here the input parameter rest_type specifies a branch run. The branch file (/DOE/.../r1981-01-01) must be specified. Its prefix, "mss:", indicates that it is found on NCAR's MSS. This initial condition file was created by a previous simulation. The date from the branch file will be used (apparently 1981 Jan 1). The simulation will stop after advancing 31 days (1981 Feb 1). Restart files will be created every 10 days relative to the start of the run. The absence of a archival device prefix on mss_dir results in the same behavior as a "cp:" prefix, i.e. history and restart files will be archived by being copied to the NFS mounted directory specified by mss_dir.

# 4 Output Data

The coupler outputs three types of data: standard out (stdout) diagnostic data, restart files, and diagnostic history files.

## 4.1 Stdout Data

Stdout output consists mostly of brief messages that indicate how the simulation is progressing and whether any error conditions have been detected. Stdout also contains a record of the values of all Coupler input parameters. If global diagnostics have been activated (see the diag_freq namelist parameter), stdout will also contain some diagnostic information, specifically global averages and time averages of various flux fields flowing through the Coupler.

Exactly where the Coupler's stdout (and stderr, standard error) shows up is determined outside of the Coupler source code and executable. Certain environment variables found in the CCSM2.0.1 run scripts are used to determine where the Coupler will execute (its current working directory, or "cwd") and also to redirect stdin and stdout -- these env variables determine where stdout text will end up. Normally the run script arranges for stdout text to show up in a "log" file in the Coupler's execution directory (current working directory) while the simulation is in progress, and then later, after the simulation ends, the run script moves this log file to an archival location.
See the CCSM2.0.1 User's Guide for details.

## 4.2 Restart Files

Restart files are in a machine dependent binary format, written using simple, standard fortran write statements. Restart files provide the Coupler with all the IC data necessary to do an "exact restart" of a previous simulation.
"Exact restart" means stopping and restarting a simulation while preserving, bit-for-bit, the results that would have been created if the simulation had not been stopped and restarted.

"Continuation" or "branch" runs can do exact restarts, and thus require a restart file. For "startup" or "hybrid" runs, the Coupler does not need or use a restart file, hence these types of runs cannot do exact restarts.
See the CCSM2.0.1 User's Guide for more information about these runtypes.

Normally there is no need to examine the contents of a restart file. All fields found in the restart file can be saved into history files, which are machine independent netCDF files.

## 4.3 History Files

History File Format

NetCDF (network Common Data Form) was chosen as the history data format because many common visualization tools already exist that handle this data format. NetCDF is an interface for array-oriented data access and a library that provides an implementation of the interface. The netCDF library also defines a machine independent format for representing scientific data. Together the interface, library, and format support the creation, access, and sharing of scientific data. The netCDF software was developed at the Unidata Program Center in Boulder, Colorado. See http://www.unidata.ucar.edu/packages/netcdf.

History File Content

Because netCDF files are self-describing, the most complete and accurate description of the contents of a Coupler history file (or any netCDF file) will always come from the netCDF data file itself. The netCDF tool "ncdump" will generate the CDL text representation of a netCDF file on standard output, optionally excluding some or all of the variable data in the output.

The Coupler's netCDF files conform to the CF 1.0 format.
The CF conventions generalize and extend the COARDS conventions.
See http://www.cgd.ucar.edu/cms/eaton/cf-metadata.

Three types of data are found in Coupler netCDF history data files:
global attributes, domain data, and two dimensional state and flux data.

(1) Global attributes

This is text information, including the case name corresponding to the history data, and the date the data was created.

(2) Model domain data

This includes the spatial coordinates of the grid cells as well as the cell areas and a domain mask for each surface model. Each model has two sets of latitude and longitude coordinates, one corresponding to grid cell "centers", xc and yc, and one corresponding to grid cell "vertices", xv and yv. Each cell is defined by four vertices which describe a quadrilateral. Grid cell "centers" lie within this polygon, typically near its center.

Data is indexed by i, j, and k: xc(i,j), yc(i,j) and xv(i,j,k), yc(i,j,k),
because the underlying data structures are 2D arrays indexed by (i,j),
and because all cells have four vertices, indexed by k (k=1,2,3,4).
Note that latitude and longitude are a function of both i and j, which is general enough to describe, for example, shifted pole grids.

A state variable S(i,j) is understood to be a point value located at the center of cell (i,j). A flux field F(i,j) can be thought of as a point value located at the cell center, but more accurately it is an area average value that applies uniformly over the entire grid cell.

Five sets of coordinate arrays are found in the history data:

* xc_a(i,j), yc_a(i,j), xv_a(i,j,k), and yv_a(i,j,k) are the center and vertex coordinates for the atmosphere model grid.
* xc_i(i,j), yc_i(i,j), xv_i(i,j,k), and yv_i(i,j,k) are the center and vertex coordinates for the sea-ice model grid.
* xc_l(i,j), yc_l(i,j), xv_l(i,j,k), and yv_l(i,j,k) are the center and vertex coordinates for the land model grid.
* xc_r(i,j), yc_r(i,j), xv_r(i,j,k), and yv_r(i,j,k) are the center and vertex coordinates for the land model's runoff grid.
* xc_o(i,j), yc_o(i,j), xv_o(i,j,k), and yv_o(i,j,k) are the center and vertex coordinates for the ocean model grid.

Each surface model (land, land runoff, sea-ice, and ocean) also has a corresponding domain mask. The domain mask is an integer array such that if mask(i,j) /= 0, then the indices (i,j) correspond to a grid cell that is in the active model domain, i.e., S(i,j) is a valid model state and F(i,j) is a valid flux value. Conversely, if mask(i,j) = 0, then S(i,j) and F(i,j) are undefined.

Three masks are found in the history data:

* mask_a(i,j) is the atmosphere model domain mask.
* mask_l(i,j) is the land model domain mask.
* mask_r(i,j) is the land model's runoff domain mask.
* mask_i(i,j) is the ice  model domain mask.
* mask_o(i,j) is the ocean model domain mask.

While there is an atmosphere domain mask, it is assumed that it is non-zero everywhere, ie. all atmosphere data points are assumed to be valid points within the atmosphere domain.

(3) Two dimensional state and flux data

This includes model state variables, component flux fields, and merged input flux fields. The naming convention for two dimensional variables follows the convention introduced in the pseudo-code section of this document. Some examples of state variable names are:

* Sa_a_t is an atmosphere state variable, is on the atmosphere grid, and the variable is temperature.
* Sa_a_u is an atmosphere state variable, is on the atmosphere grid, and the variable is zonal velocity.
* Sa_o_u is an atmosphere state variable, is on the ocean grid, and the variable is zonal velocity.
* So_o_t is an ocean state variable, is on the ocean grid, and the variable is temperature.

```
Some examples of flux field variable names are:

   * Faoa_a_prec is an atmosphere/ocean flux field computed by the atmosphere,
 is on the atmosphere grid, and the field is precipitation.
   * Faoa_o_prec is an atmosphere/ocean flux field computed by the atmosphere,
 is on the ocean grid, and the field is precipitation.
   * Faoc_o_taux is an atmosphere/ocean flux field computed by the Coupler,
                 is on the ocean grid, and the field is zonal wind stress.
   * Faoc_a_taux is an atmosphere/ocean flux field computed by the Coupler,
                 is on the atmosphere grid, and the field is zonal wind stress.

Each variable in the netCDF history file has long_name and units attributes
which further describe the variable. Also see the Data Exchanged section
of this document.
```

# 5 Source Code Maintenance

## 5.1 Obtaining Source Code

Coupler source code (cpl5) is available as part of the CCSM 2.0.1 distribution at
http://www.ccsm.ucar.edu/models/ . This distribution includes the source code for all CCSM component models. Documentation for other CCSM component models, as well as input data for running the models, is also available at this site.

## 5.2 Coupler Source Code

```
The Coupler source code is written almost entirely using standard Fortran 90.

The Coupler source code was developed using the CVS revision control system, but
only one "tagged" version of the Coupler is available within any source code
distribution.  This information can be used to identify the code contained in a
particular distribution.

A user may wish to modify their copy of the Coupler source code. If one
wishes to modify the Coupler source code, it is strongly recommended that
one first study the "pseudo-code" section of this document in conjunction with
studying the source code file "main.F90." This should provide a good top-down
overview of how the Coupler works, a necessary prerequisite for successful code
modification.
```

### 5.3 Shared Source Code

The Coupler source code itself (found in .../models/cpl/cpl5/ ) is incomplete and cannot be compiled (due to missing subroutines) unless it is compiled along with "CCSM shared code" (found in .../models/csm_share/ ). This shared code is an un-compiled library of support routines.

The Coupler source code itself has no machine dependencies, although the CCSM shared code does have some machine dependencies. One function of the shared code is to collect and isolate machine Dependant code, and to provide CCSM component models with machine-independent wrappers to such code.

Another function of the shared code is to provide a mechanism for the various component models to be consistent with one another, for example, to use an identical value for pi or for the latent heat of fusion. Similarly, the shared code contains a library routine for calculating a solar angle, so that all component models can be consistent in their solar angle calculations.

### 5.4 Shared Build Environment

The CCSM distribution includes a shared build environment which includes a makefile (a GNU makefile), a variety of machine-dependent makefile macro files and a dependency generator. This common build environment is used to build all CCSM components including the Coupler. The build environment is found in .../models/bld subdirectory of the CCSM source code distribution.

The makefile, which requires the use of gnu-make, is machine independent, but it "includes" (a standard make functionality) a machine-dependent macros definition file. Several macros files are included in the distribution, but because such macro definitions are typically very machine and site specific, it is expected that end users will need to create a new macros definition file for their site.

Also part of the build environment is a dependency generator. This is written in standard c, and thus is compiled with the standard Unix cc command. The dependency generator is particularly useful when hacking code, either by modifying some files or adding new ones.

# 6 Pseudo-Code

To understand the details of the Coupler's functionality, it is useful to have a basic understanding of the computer code that is the Coupler. To that end, pseudo-code is shown below. If one wishes to modify the Coupler source code, it is strongly recommended that one first study this section in conjunction with studying the source code file, main.F90. This should provide a good overview of how the Coupler works; a necessary pre-requisite for successful code modification. The actual source code is 99.9% pure Fortran 90.

## 6.1 Variable Naming Convention

A variable naming convention has been adopted to help organize and identify the literally hundreds of state and flux fields managed by the Coupler. Understanding this naming convention is a prerequisite for understanding the pseudo-code below.

### State Variables

Model state variables are denoted Sx, where x denotes the corresponding component model:

```
   * Sa = atm state variables, e.g. atm wind velocity
   * Si = ice state variables, e.g. ice temperature
   * Sl = lnd state variables, e.g. lnd albedo
   * So = ocn state variables, e.g. ocn SST
   * Ss = all surface states merged together, e.g. global surface temperature
```

Next, a suffix _x indicates what model grid the states reside on:

```
   * Sa_a: atm states on atm grid
   * Sa_o: atm states on ocn grid
```

### Flux Fields

Input flux means fluxes given by the Coupler to a model. Output flux means fluxes computed within a model and given to the Coupler. All input fluxes for one model were either computed by the Coupler or were the output flux of another model. In general, a given flux field between any two component models may have been computed in one of three places: within either of the two models or within the Coupler.

One function of the Coupler is to gather, merge, sum, and/or time-average the various component flux fields from various sources and form a set of complete input fluxes for each component model. This gathering and merging

process will generally involve mapping flux fields between various model
grids and combining like fields from several grids onto one grid. A
summation might be required, e.g., net heat flux = solar + latent + sensible
+ longwave. Also, for some flux fields the Coupler might be required to form
time-averaged quantities. Thus component fluxes are mapped, merged, summed,
and/or time-averaged by the Coupler in order to form complete input fluxes
for the models.

Flux fields are denoted Fxyz, where xy denotes the two model between which a
quantity is being fluxed, and z denotes the model which computed the flux
(i.e. it is an output flux of model z).

Component fluxes that are gathered, merged, summed, and/or time--averaged to
form the complete input fluxes are:

```
   * Faia = atm/ice flux, computed by atm, e.g. precipitation
   * Faii = atm/ice flux, computed by ice, e.g. sensible heat flux
   * Fala = atm/lnd flux, computed by atm, e.g. precipitation
   * Fall = atm/lnd flux, computed by lnd, e.g. sensible heat flux
   * Faoc = atm/ocn flux, computed by cpl, e.g. momentum flux
   * Faoa = atm/ocn flux, computed by atm, e.g. precipitation
   * Fioo = ice/ocn flux, computed by ocn, e.g. ice formed within the ocn
   * Fioi = ice/ocn flux, computed by ice, e.g. penetrating shortwave radiation
   * Flol = lnd/ocn flux, computed by lnd, e.g. river runoff
```

Complete input fluxes (an "a" prefix denotes a daily average):

```
   *  Faxx = all atm input fluxes: a map/merge/sum of: Faoc, Faii, Fall
   *  Flxx = all lnd input fluxes: a map/merge/sum of: Fala
   *  Foxx = all ocn input fluxes: a map/merge/sum of: Faoc, Faoa, Fioi, Flol
   * aFoxx = a time average of: Foxx
   *  Fixx = all ice input fluxes: a map/merge/sum of: Faia, Fioo
```

Finally, just like the state variables above, a suffix _x indicates what
model grid the fluxes reside on:

```
   * Faoa_a: atm/ocn fluxes, computed by the atm, on the atm grid
   * Faoa_o: atm/ocn fluxes, computed by the atm, on the ocn grid
```

## Domain Maps

Mapping between domains is implemented by matrix multiplies. Generally there
are different maps for state fields vs. flux fields, as state fields maps are
generally smoother (eg. bilinear) whereas flux field maps must be conservative
(eg. area averaging, aka Riemann sum integrals). In the case of mapping between

identical domains, the map would be the identity map. The necessary mapping matrices are:

```
   * map_Fa2i: map for fluxes, atm -> ice
   * map_Sa2i: map for states, atm -> ice
   * map_Fa2l: map for fluxes, atm -> lnd
   * map_Sa2l: map for states, atm -> lnd
   * map_Fa2o: map for fluxes, atm -> ocn
   * map_Sa2o: map for states, atm -> ocn
   * map_Fi2a: map for fluxes, ice -> atm
   * map_Si2a: map for states, ice -> atm
     etc.
```

## 6.2 Main Program

Notice that the Coupler code is configured for a particular time coordination scheme. Model time coordination involves two communication intervals, with the longer interval being an integer multiple of the shorter interval. The atmosphere, ice, and land models communicate once per short interval while the ocean model communicates once per long interval. Also, one day (24 hours) is an integer multiple of the longer communication interval. Typically the longer interval is exactly one day, while the shorter interval is several hours or less. While this configuration is hard-coded within the Coupler, its modular design facilitates code modifications to implement alternate configurations. A variety of time coordination schemes can be, and have been, implemented by rearranging subroutine calls at the highest level (within the main program), requiring a minimal amount of code modifications or new code.

```
PROGRAM main

   call initial       ! read input namelist

   call msg_atm_init  ! exchange spatial & temporal grid info with atm model
   call msg_ice_init  ! exchange spatial & temporal grid info with ice model
   call msg_lnd_init  ! exchange spatial & temporal grid info with lnd model
   call msg_ocn_init  ! exchange spatial & temporal grid info with ocn model

   !--- read restart file ---
   call restart ('read' , Sa_a , Faia_a , Fala_a , Faoa_a ,
   &                      Sl_l , Fall_l , Falo_l ,
   &                      Si_i , Faii_i , Fioi_i ,
   &                      So_o , Fioo_o ,aFoxx_o )

   m = ncpl_a/ncpl_o  ! determine number of atm messages per ocn message
   DO WHILE ( current_date < stop_date )

      DO n=0,ncpl_a-1

         !--- send tavg inputs to ocn, start new tavg summation ---
         if ( mod(n,m) == 0 ) then
            call msg_ocn_send (aFoxx_o)
            call tavg-zero    (aFoxx_o) ! start new time average
         end if

         !--- map atm states/fluxes to lnd, send msg to lnd ---
         call map (map_Sa2l,   Sa_a ,  Sa_l )
         call map (map_Fa2l, Fala_a ,Fala_l )
!lnds    call msg_lnd_send ( Fala_l ,  Sa_l  )

         !--- map atm states/fluxes to ice grid ---
         call map (map_Sa2i,   Sa_a ,   Sa_i )
         call map (map_Fa2i, Faia_a , Faia_i )

         !--- send msg to ice ---
!ices    call msg_ice_send ( Sa_i , So_i, Faia_i, Fioo_i )

         !--- prepare instantaneous ocn inputs ---
         call map (map_Sa2o,   Sa_a ,   Sa_o )
         call map (map_Fa2o, Faoa_a , Faoa_o )
         call map (map_Fl2o, Flol_l , Flol_o )
         call flux_solar ( Faoa_o, So_o, Faoc_o ) ! compute net short-wave

         !--- "send" msg to ocn (add instantaneous data to t-avg) ---
!ocns    call mrg_ocn  ( Faoa_o,  Faoc_o, Fioi_o, Flol_o, Foxx_o)
         call tavg-sum ( Foxx_o, aFoxx_o)

         !--- compute ocean albedos and Faoc fluxes ---
!ocnr    call flux_albedo ( So_o ) ! update ocn state ~ "receive"
         call flux_ao     ( Sa_o, So_o, Faoc_o )

         !--- map ocn states/fluxes to atm ---
         call map( map_Fo2a, Faoc_o, Faoc_a )
         call map( map_So2a,   So_o,   So_a )

!icer    !--- receive ice outputs ---
         call msg_ice_recv (Si_i ,Faii_i, Fioi_i )

         !--- map ice states/fluxes to atm & ocn grid ---
         call map (map_Fi2a, Faii_i, Faii_a )
         call map (map_Si2a,   Si_i,   Si_a )
         call map (map_Fi2o, Fioi_i, Fioi_o )
         call map (map_Si2o,   Si_i,   Si_o )

!lndr    !--- recv new lnd data ---
         call msg_lnd_recv (Sl_l, Fall_l, Flol_l )

         !--- map lnd states & fluxes to atm ---
         call map (map_Fl2a, Fall_l, Fall_a )
         call map (map_Sl2a,   Sl_l,   Sl_a )

         !--- merge atm states & fluxes ---
         call mrg_atm (Faii_a, Fall_a, Faoc_a, Faxx_a)
         call mrg_atm (  Si_a,   Sl_a,   So_a,   Ss_a)

!atms    !--- send data to atm ---
         call msg_atm_send (Faxx_a, Ss_a )

         !--- add data to history file ---
         call history ()

         if ( mod(n+1,m) == 0 ) then
            !--- receive ocn outputs ---
            call msg_ocn_recv (So_o , Fioo_o)

            !--- map ocn to ice ---
            call map (map_So2i,   So_o,   So_i )
            call map (map_Fo2i, Fioo_o, Fioo_i )

            !--- form tavg of ocn inputs summation---
            call tavg-avg(aFoxx_o,m)
         end if

!atmr    !--- recv msg from atm ---
         call msg_atm_recv (Sa_a, Faia_a, Fala_a, Faoa_a)

         !--- cpl, atm, ice, lnd have advanced part of one day ---
         sec = sec + 86400/ncpl_a
         call control()     ! reset stop, rest, hist control flags
      END DO

      !--- cpl & component models have advanced one day ---
      if (sec >= 86400) then
         eday = eday + 1  ! increment number of elapsed days
         sec  = 0
      end if
      call tcheck()    ! verify time coordination
      call control()   ! reset stop, rest, hist control flags

      !--- make a cpl restart file ---
      call restart ('write', Sa_a , Faia_a , Fala_a , Faoa_a ,
      &                      Sl_l , Fall_l , Falo_l ,
      &                      Si_i , Faii_i , Fioi_i ,
      &                      So_o , Fioo_o ,aFoxx_o )

   END DO WHILE

999 CONTINUE

   !--- send final msg to models & disconnect from msg-passing ---
   call msg_atm_final ()
   call msg_ice_final ()
   call msg_ocn_final ()
   call msg_lnd_final ()
   call msg_cpl_disconnect ()

STOP/END OF PROGRAM
```

# 7 Data Exchanged with Component Models

Each component model exchanges data with the Coupler only. Component models have no direct connection with each other -- all data is routed through the Coupler. Most data is in the form of 2D fields. This data is accompanied by certain timing and control information (arrays of scalar real or integer values), such as the current simulation data and time.

## 7.1 Units Convention

All data exchanged conforms to this units convention:

```
o Sign convention:
     positive value <=> downward flux
o Unit convention:
     temperature   ~ Kelvin
     salt          ~ g/kg
     velocity      ~ m/s
     pressure      ~ N/m^2 = Pa
```

| | | |
|---|---|---|
| humidity | ~ | kg/kg |
| air density | ~ | kg/m^3 |
| momentum flux | ~ | N/m^2 |
| heat flux | ~ | W/m^2 |
| water flux | ~ | (kg/s)/m^2 |
| salt flux | ~ | (kg/s)/m^2 |
| coordinates | ~ | degrees north or east |
| area | ~ | radians^2 |
| domain mask | ~ | 0 <=> an inactive grid cell |

## 7.2 Time Invariant Data

This section provides a list of the time invariant data exchanged between the Coupler and each component model. Generally this data is the "domain" data: coordinate arrays, domain mask, cell areas, etc. It is assumed that the domain of all models is represented by a 2D array (although not necessarily a latitude/longitude grid).

Data sent to Coupler

domain data

```
* grid cell's center coordinates, zonal      (degrees north)
* grid cell's center coordinates, meridional (degrees east)
* grid cell's four vertex coordinates, zonal      (degrees north)
* grid cell's four vertex coordinates, meridional (degrees east)
* grid cell area (radians squared)
* grid cell domain mask ( 0 <=> not in active domain)
* ni,nj: the dimensions of the underlying 2D array data structure
```

time coordination data

* ncpl: number of times per day the component will communicate (exchange data) with the Coupler.

other information

* IC flag: indicates whether the Coupler should use model IC's contained on the Coupler's restart file or IC's in the initial message sent from the component model.

Data sent to Component Models

time coordination data

* date, seconds: the exact time the Coupler will start the simulation from.

## 7.3 Time Variant Data

This section provides a list of the time-evolving data sent exchanged between the Coupler and each component model. Generally this is state, flux, and diagnostic quantities.

Each component model provides the Coupler with a set of output fields. Output fields from a model include output states (which can be used by another component to compute fluxes) and output fluxes (fluxes that were computed within the model and which need to be exchanged with another component model.

The Coupler provides each component model with input fields. Input fields sent to a model include input states (the state variables of other models, which are needed to do a flux calculation) and input fluxes (a forcing fields computed by some other component).

Flux fields sent to or from the Coupler are understood to apply over the communication interval beginning when the data was received and ending when the next message is received. The component models must insure that fluxes sent to the Coupler are computed with this in mind -- failure to do so may result in the non-conservation of fluxes. For example, if the atmosphere component communicates with the Coupler once per hour, but takes three internal time steps per hour, then the precipitation (water flux) sent to the Coupler should be the average precipitation over an hour (the average precipitation over three internal time steps). Similarly, if the ocean component has a communication interval of one day, but takes 50 internal time steps per day, then the precipitation flux field it receives from the Coupler should be applied as ocean boundary condition forcing for all 50 time steps during the next communication interval.

### 7.3.1 Atmosphere Model

```
Data sent to Coupler

     states
   * layer height (m)
   * zonal      velocity (m/s)
   * meridional velocity (m/s)
   * temperature (Kelvin)
   * potential temperature (Kelvin)
   * pressure (Pa)
   * equivalent sea level pressure (Pa)
   * specific humidity (kg/kg)
   * density humidity (kg/m^3)

     fluxes
   * precipitation: liquid, convective  ((kg/s)/m^2)
   * precipitation: liquid, large-scale ((kg/s)/m^2)
   * precipitation: frozen, convective  ((kg/s)/m^2)
   * precipitation: frozen, large-scale ((kg/s)/m^2)
   * longwave  radiation, downward                        (W/m^2)
   * shortwave radiation: downward, visible     , direct  (W/m^2)
   * shortwave radiation: downward, near-infrared, direct  (W/m^2)
   * shortwave radiation: downward, visible     , diffuse (W/m^2)
   * shortwave radiation: downward, near-infrared, diffuse (W/m^2)

     diagnostic quantities
   * net shortwave radiation (W/m^2)

Data received from Coupler

     states
   * albedo: visible      , direct
   * albedo: near-infrared, direct
   * albedo: visible      , diffuse
   * albedo: near-infrared, diffuse
   * surface temperature (Kelvin)
   * snow height (m)
   * ice   fraction
   * ocean fraction
   * land  fraction  (implied by ice and ocean fractions)

     fluxes
   * zonal      surface stress (N/m^2)
   * meridional surface stress (N/m^2)
   * latent heat   (W/m^2)
   * sensible heat (W/m^2)
   * longwave radiation, upward (W/m^2)
   * evaporation ((kg/s)/m^2)

     diagnostic quantities
   * 2 meter reference air temperature (Kelvin)
```

### 7.3.2 Ice Model

```
Data sent to Coupler
```

```
     states
   * ice fraction
   * surface temperature (Kelvin)
   * albedo: visible      , direct
   * albedo: near-infrared, direct
   * albedo: visible      , diffuse
   * albedo: near-infrared, diffuse

     fluxes
   * atm/ice: zonal      surface stress (N/m^2)
   * atm/ice: meridional surface stress (N/m^2)
   * atm/ice: latent heat               (W/m^2)
   * atm/ice: sensible heat             (W/m^2)
   * atm/ice: longwave radiation, upward (W/m^2)
   * atm/ice: evaporation ((kg/s)/m^2)
   * ice/ocn: penetrating shortwave radiation (W/m^2)
   * ice/ocn: ocean heat used for melting (W/m^2)
   * ice/ocn: melt water ((kg/s)/m^2)
   * ice/ocn: salt flux  ((kg/s)/m^2)
   * ice/ocn: zonal      surface stress (N/m^2)
   * ice/ocn: meridional surface stress (N/m^2)

     diagnostic quantities
   * net shortwave radiation (W/m^2)
   * 2 meter reference air temperature (Kelvin)


Data received from Coupler

     states
   * ocn: temperature (Kelvin)
   * ocn: salinity    (g/kg)
   * ocn: zonal      velocity (m/s)
   * ocn: meridional velocity (m/s)
   * atm: layer height (m)
   * atm: zonal      velocity (m/s)
   * atm: meridional velocity (m/s)
   * atm: potential temperature (Kelvin)
   * atm: temperature           (Kelvin)
   * atm: specific humidity (kg/kg)
   * atm: density (kg/m^3)

     fluxes
   * ocn: dh/dx: zonal      surface slope (m/m)
   * ocn: dh/dy: meridional surface slope (m/m)
   * ocn: Q>0: heat of fusion    (W/m^2), or
```

```
        Q<0: melting potential (W/m^2)
   * atm: shortwave radiation: downward, visible      , direct  (W/m^2)
   * atm: shortwave radiation: downward, near-infrared, direct  (W/m^2)
   * atm: shortwave radiation: downward, visible      , diffuse (W/m^2)
   * atm: shortwave radiation: downward, near-infrared, diffuse (W/m^2)
   * atm: longwave  radiation, downward (W/m^2)
   * atm: precipitation: liquid ((kg/s)/m^2)
   * atm: precipitation: frozen ((kg/s)/m^2)
```

### 7.3.3 Land Model

```
Data sent to Coupler

     states
   * surface temperature (Kelvin)
   * albedo: visible      , direct
   * albedo: near-infrared, direct
   * albedo: visible      , diffuse
   * albedo: near-infrared, diffuse
   * snow depth (m)

     fluxes
   * zonal      surface stress  (N/m^2)
   * meridional surface stress  (N/m^2)
   * latent heat                (W/m^2)
   * sensible heat              (W/m^2)
   * longwave radiation, upward (W/m^2)
   * evaporation    ((kg/s)/m^2)
   * coastal runoff ((kg/s)/m^2)

     diagnostic quantities
   * 2 meter reference air temperature (Kelvin)

Data received from Coupler

     states
   * atm layer height (m)
   * atm zonal      velocity (m/s)
   * atm meridional velocity (m/s)
   * atm potential temperature (Kelvin)
   * atm specific humidity (kg/kg)
   * atm pressure (Pa)
   * atm temperature (Kelvin)
```

```
     fluxes
   * precipitation: liquid, convective  ((kg/s)/m^2)
   * precipitation: liquid, large-scale ((kg/s)/m^2)
   * precipitation: frozen, convective  ((kg/s)/m^2)
   * precipitation: frozen, large-scale ((kg/s)/m^2)
   * longwave  radiation, downward (W/m^2)
   * shortwave radiation: downward, visible      , direct  (W/m^2)
   * shortwave radiation: downward, near-infrared, direct  (W/m^2)
   * shortwave radiation: downward, visible      , diffuse (W/m^2)
   * shortwave radiation: downward, near-infrared, diffuse (W/m^2)
```

### 7.3.4 Ocean Model

```
Data sent to Coupler

     states
   * surface temperature (Kelvin)
   * salinity (g/kg)
   * zonal      velocity (m/s)
   * meridional velocity (m/s)

     fluxes
   * dh/dx: zonal      surface slope (m/m)
   * dh/dy: meridional surface slope (m/m)
   * Q>0: heat of fusion    (W/m^2), or
     Q<0: melting potential (W/m^2)

Data received From Coupler

     states
   * equivalent sea level pressure (Pa)
   * ice fraction

     fluxes
   * zonal      surface stress (N/m^2)
   * meridional surface stress (N/m^2)
   * shortwave radiation, net  (W/m^2)
   * latent heat                  (W/m^2)  (note: derived from evaporation)
   * sensible heat                (W/m^2)
   * longwave radiation, upward   (W/m^2)
   * longwave radiation, downward (W/m^2)
   * ocean heat used for melting  (W/m^2)
   * salt flux  ((kg/s)/m^2)
   * precipitation: rain + snow  ((kg/s)/m^2)
```

```
  * evaporation                 ((kg/s)/m^2)
  * melt water                  ((kg/s)/m^2)
  * coastal runoff              ((kg/s)/m^2)
```

# 8 Flux Calculations

The Coupler is required to compute certain fluxes and other quantities related to fluxes, such computing the ocean's surface albedo. These calculations are described below.

## 8.1 Atmosphere/Ocean Fluxes

### 8.1.1 General Expressions

The fluxes across the interface are calculated from bulk formulae and general expressions are

$$\begin{aligned}
\vec{\tau} &= \rho_A \; u^{*2} \; \Delta\vec{U} \; |\Delta\vec{U}|^{-1} \\
E &= \rho_A \; u^* \; Q^* \\
H &= \rho_A \; Cp_A \; u^* \; \theta^* \\
L\uparrow &= -\,\sigma \; T^4 \quad \simeq \quad -\,\epsilon\,\sigma \; T^4 \quad + \quad \alpha^L \; L\downarrow,
\end{aligned} \tag{8.1}$$

where the turbulent velocity scales are given by

$$\begin{aligned}
u^* &= CD^{1/2} \; |\Delta\vec{U}| \\
Q^* &= CE \; |\Delta\vec{U}| \; (\Delta q) \; u^{*-1} \\
\theta^* &= CH \; |\Delta\vec{U}| \; (\Delta\theta) \; u^{*-1},
\end{aligned} \tag{8.2}$$

where $\rho_A$ is atmospheric surface density, $Cp_A$ is the specific heat, $\sigma = 5.67 \times 10^{-8}\mathrm{W/m^2/K^4}$ is the Stefan-Boltzmann constant, $\epsilon$ is the emissivity of the interface, and $\alpha^L$ is the surface albedo for incident longwave radiation, $L\downarrow$. In (8.2) the differences $\Delta\vec{U}$, $\Delta q$ and $\Delta\theta$ are defined at each interface in accord with the convention of fluxes being positive down. The reflected downward incident longwave radiation is simply accounted for by assuming an emissivity, $\epsilon = 1$, and the water surface albedo for incident longwave radiation, $\alpha^L = 0.0$.

The transfer coefficients in (8.2), shifted to a height, $Z$, and considering the appropriate stability parameter, $\zeta$, are :

(8.3)

$$\begin{aligned}
CD &= \kappa^2 \left[ ln \left( \frac{Z}{Z^o} \right) - \psi_m \right]^{-2} \\
CE &= \kappa^2 \left[ ln \left( \frac{Z}{Z^o} \right) - \psi_m \right]^{-1} \left[ ln \left( \frac{Z}{Z^e} \right) - \psi_s \right]^{-1} \\
CH &= \kappa^2 \left[ ln \left( \frac{Z}{Z^o} \right) - \psi_m \right]^{-1} \left[ ln \left( \frac{Z}{Z^h} \right) - \psi_s \right]^{-1} ,
\end{aligned}$$

where $\kappa = 0.4$ is von Karman's constant and the integrated flux profiles, $\psi_m$ for momentum and $\psi_s$ for scalars, are functions of the stability parameter, $\zeta$. These functions as used in the coupler are:

$$\begin{aligned}
\psi_m(\zeta) &= \psi_s(\zeta) = -5\zeta & \zeta > 0 \\
\psi_m(\zeta) &= 2ln[0.5(1+X)] + ln[0.5(1+X^2)] - 2tan^{-1}X + 0.5\pi & \zeta < 0 \\
\psi_s(\zeta) &= 2ln[0.5(1+X^2)] & \zeta < 0 \\
X &= (1-16\zeta)^{1/4}
\end{aligned}$$

Above the atmospheric interfaces $i = 1,\ 2$ and 3 the stability parameter

$$\zeta = \frac{\kappa\ g\ Z_A}{u^{*2}} \left( \frac{\theta^*}{\theta_v} + \frac{Q^*}{(Z_v^{-1} + q_A)} \right) ,$$

where virtual potential temperature is computed as $\theta_v = \theta_A(1 + Z_v q_A)$, $q_A$ and $\theta_A$ are the lowest level atmospheric humidity, and potential temperature, respectively, and $Z_v = (\varrho(water)/\varrho(air)) - 1 = 0.606$.

In addition to surface fluxes, the atmospheric model requires effective surface albedos for both direct $\alpha(dir)$, and diffuse, $\alpha(dif)$, radiation at each wavelength. They are used in a single call to the computationally demanding atmospheric radiation routines. This call gives downward atmospheric albedo for diffuse radiation, $\alpha_a(dif)$. If direct and diffuse albedos, $\alpha^m(dir)$ and $\alpha^m(dif)$ for $m = 1, M$ different surfaces below an atmospheric grid cell are known, the multiple scattering coefficients, $R^m = (1 - \alpha_a(dif)\,\alpha^m(dif))^{-1}$ are computed, and with $f^m$ the fractional coverage of each surface type, then the albedos are given by

$$\begin{aligned}
\alpha(dif) &= (1 - F_2) \,/\, (1 - F_2 \alpha_a(dif)) \\
\alpha(dir) &= F_1 \, (1 - \alpha_a(dif)\, \alpha(dif))
\end{aligned}$$

$$F_1 = \sum_{m=1}^{M} f^m\, R^m\, \alpha^m(dir)$$

$$F_2 = \sum_{m=1}^{M} f^m\, R^m\, (1 - \alpha^m(dif))$$

Use of these albedos ensures that the solar radiation exiting the bottom of the atmosphere is identical to the sum of $M$ radiation calculations, each using an $\alpha^m(dif)$ and $\alpha^m(dir)$. At present, however, the albedo calculations are greatly simplified by assuming $\alpha_a(dif) = 0$, such that $R^m = 1$. This albedo then does not need to be passed from the atmosphere to the coupler, and the coupler simply computes the effective albedos to be passed to the atmosphere as

$$\begin{aligned}\alpha(dif) &= \sum_{m=1}^{M} f^m\,\alpha^m(dif)\\ \alpha(dir) &= \sum_{m=1}^{M} f^m\,\alpha^m(dir)\end{aligned}$$

In general the partition of radiation among the $M$ surfaces is a function of $R^m$, $\alpha^m(dif)$, and $\alpha^m(dir)$, and hence of wavelength. Hence, correct partitioning would need to be performed for each wavelength band within the radiation transfer code of the atmospheric model. This procedure is greatly simplified by partitioning the net solar radiation within the Coupler.

### 8.1.2 Specific Expressions

At the atmosphere-ocean interface the near-surface air is also assumed to be saturated,

$$q = 0.98\ \rho_A^{-1}\ C_5\ \exp(C_6/T),$$

where the sea surface temperature is $T = SST$, $C_5 = 640380.0$ kg/m$^3$ and $C_6 = -5107.4$ K, and the factor 0.98 accounts for the salinity of the ocean. The differences (8.2) become

$$\begin{aligned}\Delta\vec{U} &= \vec{U}_A - \vec{U}\\ \Delta q &= q_A - q\\ \Delta\theta &= \theta_A - T,\end{aligned}$$

where the surface current, $\vec{U}$, is presently assumed to be negligible.

The roughness length for momentum, $Z^o$ in meters, is a function of the atmospheric wind at 10 meters height, $U_{10}$:

$$Z^o = 10\ \exp - \left[\kappa\left(\frac{C_1}{U_{10}} + C_2 + C_3U_{10}\right)^{-1/2}\right],$$

where $C_1 = 0.0027$ m/s, $C_2 = 0.000142$, and $C_3 = 0.0000764$ m$^{-1}$s. The corresponding drag coefficient at 10m height and neutral stability is

$$C_{10}^N = C_1\,U_{10}^{-1} + C_2 + C_3\,U_{10}\ .$$

The roughness length for heat, $Z^h$, is a function of stability, and for evaporation, $Z^e$, is a different constant:

$$\begin{aligned} Z^h &= 2.2 \times 10^{-9} m \qquad \zeta > 0 \\ &= 4.9 \times 10^{-5} m \qquad \zeta \leq 0 \\ Z^e &= 9.5 \times 10^{-5} m. \end{aligned}$$

Since $\zeta$ is itself a function of the turbulent scales (8.2), and hence the fluxes, an iterative procedure is generally required to solve (8.1). First, $\zeta$ is set incrementally greater than zero when the air–sea temperature difference suggests stable stratification, otherwise it is set to zero. In either case, $\psi_m = \psi_s = 0$, and the initial transfer coefficients are then found from the roughness lengths at this $\zeta$ and $U_{10} = U_A$. As with sea–ice, these coefficients are used to approximate the initial flux scales (8.2) and the first iteration begins with an updated $\zeta$ and calculations of $\psi_m$ and $\psi_s$. The wind speed, $U_A$, is then shifted to its equivalent neutral value at 10m height :

$$U_{10} = U_A \ \left(1 + \frac{\sqrt{C_{10}^N}}{\kappa} ln(\frac{Z_A}{10} - \psi_m(\zeta))\ \right)^{-1} \quad .$$

This wind speed is used to update the transfer coefficients and hence the flux scales. The second and final iteration begins with another update of $\zeta$. The final flux scales then give the fluxes calculated by (8.1).

## 8.2 Surface Albedo and Net Absorbed Solar Radiation

For the ice, land, and ocean components there are four surface albedos that are used by the atmosphere component to compute four corresponding components of downward shortwave. Subsequently, the four albedos, together with the four downward shortwave fields, are used to compute net absorbed shortwave flux from the atmosphere to the surface components:

$$SW_{net} = \sum_{m=1}^{4} SW_{down}^m \ (1 - \alpha^m) \quad ,$$

where $m = 1, ..., 4$ corresponds to near-infrared/diffuse, visible/diffuse, near-infrared/direct, and visible/direct shortwave components.

The ice and land components each compute their own surface albedos, and subsequently, give the downward shortwave fields, compute their own net absorbed shortwave radiation. For the ocean component, it is the Coupler that computes the ocean surface albedo and subsequently computes net absorbed shortwave radiation. The Coupler then sends the net absorbed shortwave field to the ocean component.

### 8.2.1 Land Surface Albedos

The land surface albedos are computed by the land component and passed on to the atmosphere component. These albedos are not altered by the Coupler in any way. Note that the atmosphere component may be an active model computing downward shortwave once per hour, or it may be a data model feeding the Coupler a daily average downward shortwave. It is the user's responsibility to ensure that the albedos the land model sends are appropriate considering what type of downward shortwave fields atmosphere component is providing.

### 8.2.2 Ice Surface Albedo

The ice surface albedos are computed by the ice component and passed through the Coupler and on to the atmosphere component. These albedos are "60 degree reference albedos" that have no diurnal cycle. Based on an input namelist variable, "flx_albav" (see the namelist section of the Coupler User's Guide), the Coupler will either pass these albedos on to the atmosphere component unaltered (in which case they are considered "daily average" albedos), or impose a diurnal cycle on the albedos (in which case they are considered "instantaneous" albedos). When the Coupler does add a diurnal cycle to the ice albedo, this consists of merely setting the albedos to 1.0 on the dark side of the earth.

### 8.2.3 Ocean Surface Albedos

Unlike the ice and land components, the Coupler computes the ocean component's surface albedo. There are two ways the Coupler can compute the ocean albedo: with a diurnal cycle ("instantaneous") or without a diurnal cycle ("daily average"). An input namelist variable (the "flx_albav" namelist variable) selects which option the Coupler implements.

If the albedos are computed as daily average albedos, then all four ocean albedos are set to 0.06 everywhere, regardless of time of day or time of year.

If the albedos are computed as instantaneous albedos, then all four ocean will be set to 1.0 on the dark side of the earth, and where the solar angle is greater than zero, the albedos are set to a value which has both an annual and a diurnal cycle. Ocean albedo distinguishes between direct and diffuse radiation. The direct albedo is solar zenith angle dependent, while the diffuse is not. There is no spectral dependence of the albedo, nor dependence on surface wind speed. The expressions for both direct and diffuse albedo are taken from Briegleb et al. (1986), based on fits to observations of ocean albedo, good to within $\pm 0.3\%$. The albedo expressions are valid for open ocean, and do not include the effects of suspended hydrosols in near-surface waters.

For complete details see Briegleb, B.P., P.Minnis, V.Ramanathan, and E.Harrison, 1986. Comparison of regional clear-sky albedos inferred from satellite observations and model comparisons. Journal of Climate and Applied Meteorology, Vol. 25, pp.214-226.

## 9 List of Physical Constants

```
The CCSM code has a mechanism for facilitating the consistent use of constants
among the various CCSM components -- this is the "shared constants" module of
the CCSM "shared code"  library. The Coupler always uses shared constants
whenever possible and only defines a constant locally if no such constant is
available from the shared constants module.

The Coupler source code itself (found in .../models/cpl/cpl5/ ) is incomplete
and cannot be compiled (due to missing subroutines) unless it is compiled along
with "CCSM shared code" (found in .../models/csm_share/ ).  This shared code
includes the shared constants module.

Constants in the Coupler whose actual values come from the "shared constants"...

  pi          = SHR_CONST_PI          ! pi
  rearth      = SHR_CONST_REARTH      ! radius of earth ~ m
  g           = SHR_CONST_G           ! acceleration of gravity
  cpdair      = SHR_CONST_CPDAIR      ! specific heat of dry air
  cpwv        = SHR_CONST_CPWV        ! specific heat of water vapor
  cpvir       = cpwv/cpdair - 1.0     ! cpwv/cpdair - 1.0
  zvir        = SHR_CONST_ZVIR        ! rh2o/rair   - 1.0
  latvap      = SHR_CONST_LATVAP      ! latent heat of evaporation
  latice      = SHR_CONST_LATICE      ! latent heat of fusion
  stebol      = SHR_CONST_STEBOL      ! Stefan-Boltzmann constant
  karman      = SHR_CONST_KARMAN      ! Von Karman constant
  So          = SHR_CONST_OCN_REF_SAL ! ocn reference salinity
  Si          = SHR_CONST_ICE_REF_SAL ! ice reference salinity
  HFLXtoWFLX  = -(So-Si)/(So*latice)  ! converts heat to water for diagnostics

The actual values for the shared constants above...

  SHR_CONST_PI     = 3.14159265358979323846  ! pi
  SHR_CONST_REARTH = 6.37122e6    ! radius of earth ~ m
  SHR_CONST_G      = 9.80616      ! acceleration of gravity ~ m/s^2
  SHR_CONST_CPDAIR = 1.00464e3    ! specific heat of dry air ~ J/kg/K
  SHR_CONST_CPWV   = 1.810e3      ! specific heat of water vap ~ J/kg/K
  SHR_CONST_ZVIR   = (SHR_CONST_RWV/SHR_CONST_RDAIR)-1.0   ! RWV/RDAIR - 1.0
  SHR_CONST_LATVAP = 2.501e6      ! latent heat of evaporation ~ J/kg
  SHR_CONST_LATICE = 3.337e5      ! latent heat of fusion ~ J/kg
  SHR_CONST_STEBOL = 5.67e-8      ! Stefan-Boltzmann constant ~ W/m^2/K^4
  SHR_CONST_KARMAN = 0.4          ! Von Karman constant
  SHR_CONST_OCN_REF_SAL = 34.7    ! ocn ref salinity (psu)
  SHR_CONST_ICE_REF_SAL =  4.0    ! ice ref salinity (psu)
  SHR_CONST_RWV   =
     SHR_CONST_RGAS/SHR_CONST_MWWV    ! Water vapor gas constant ~ J/K/kg
  SHR_CONST_RDAIR =
     SHR_CONST_RGAS/SHR_CONST_MWDAIR  ! Dry air gas constant     ~ J/K/kg
  SHR_CONST_RGAS  =
     SHR_CONST_AVOGAD*SHR_CONST_BOLTZ ! Universal gas constant   ~ J/K/mole
  SHR_CONST_MWWV   = 18.016           ! molecular weight water vapor
  SHR_CONST_MWDAIR = 28.966           ! molecular weight dry air ~ kg/kmole
  SHR_CONST_BOLTZ  = 1.38065e-23      ! Boltzmann's constant ~ J/K/molecule
  SHR_CONST_AVOGAD = 6.02214e26       ! Avogadro's number ~ molecules/kmole

Constants in the Coupler whose values do not come from the "shared constants"...

  albdif = 0.06   ! 60 deg reference albedo, diffuse for ocn albedo calc
  albdir = 0.07   ! 60 deg reference albedo, direct  for ocn albedo calc
  umin   =  1.0   ! minimum wind speed (m/s)       for atm/ocn flux calc
  zref   = 10.0   ! reference height (m)           for atm/ocn flux calc
  ztref  =  2.0   ! reference height for air T (m) for atm/ocn flux calc
  spval  = 1.0e30 ! flags special value (missing value)
```

# 10 Glossary

```
combine
     Add together two or more fields that are on the same domain; e.g., latent
     heat flux plus sensible heat flux on a T42 grid with the same land mask

communication interval
     The time interval at which a component model exchanges data with the
     coupler.

concurrent execution
     When two or more coupled system components are executing at the same time.

domain
     A grid together with a domain mask.

domain mask
     A mask that indicates which grid cells are active or inactive.

grid
     Coordinate arrays without a mask, cell area, decomposition information;
     e.g., a T42 grid.
```

map
: Same as "regrid"; move a data field from one domain to another.

mask
: True/false flags associated with a specific grid; e.g., a land mask.

merge
: Take two or more fields of the same type (e.g., land, ocean, and ice surface temperatures), which are all on the same domain (e.g., all on a T42 grid, global domain), and the same number of corresponding cell fractions (fraction of land, ocean, and ice, which together sum to unity), and create one field, e.g., (merged surface temperature) =
(ice frac)*(ice T) + (ocn frac)*(ocn T) + (lnd frac)*(lnd T) .

regrid
: Same as "map"; move a data field from one domain to another.

sequential execution
: When two or more coupled system components do not execute at the same time.

timestep
: A model's internal integration increment in time. Note: the Coupler does not know a model's internal timestep, it only knows a model's communication interval.