The CCSM Climatological Data Ice Model

Version 6.0

Combined

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

Brian G. Kauffman

June 2004

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

CVS tag `$Name: $` Build date: June 17, 2004

Contents

1 Introduction

The Climatological Data Ice Model (dice) functions as the sea ice component in a CCSM configuration. Recall that a *configuration* consists of various independent *component models* (atmosphere, land, ocean, sea-ice), each connected to a *coupler*. This sea ice model component interacts with the CCSM Coupler just like any ice model would, but it is not an active model, rather, it takes ice fraction data from an input data file, fabricates surface temperature data, and sends this data to the Coupler, ignoring any forcing data received from the Coupler. Such a "dummy" ice model is useful for seeing how an active atmosphere component behaves when coupled to climatological ice extent.

Important note: When assembling a CCSM configuration, the user must carefully consider the limitations and requirements of all components and make sure that the complete set of component models will interact in a meaningful way. In particular, the user must verify that the data provided by this model is adequate for their specific application.

2 Input Datasets

The data ice model requires two input netCDF datasets: a domain dataset specifying the model grid domain and an ice fraction dataset. The ice fraction dataset contains 12 months of climatological data.

2.1 Ice Data

The ice model state variable *ice fraction* (equivalently, ice extent) is read in from a data file. On startup, this model reads in ice fraction data from a netCDF file containing 12 months of monthly mean ice fraction data on a global lat/lon grid (x(i),y(j)). A data file is provided with the CCSM distribution. The file provided can be replaced by any other ice fraction data file that is in the same format. Because netCDF files are self describing, one can query the file itself for specifics about the file format.

Note: if a CCSM configuration consists of a dummy ocean and a dummy ice model (e.g. coupled to active atmosphere and land models), normally one would coordinate the ice model's ice fraction data with the ocean model's SST data so that the SST's and ice extent are consistent. The data file provided contains a coordinated set of SST and ice fraction data.

2.2 Domain Data

On startup, this model reads in domain data from a netCDF file. Data exchanged with the coupler will be on this model domain. This file contains x & y coordinate arrays as well as a cell areas and a domain mask. The domain must be a rectilinear grid, but needn't be a latitude/longitude grid. A mask value of 0 indicates land points (i.e. not in the model's domain). The ice data described above is interpolated onto the domain specified by the domain data file – all data exchanged with the coupler is on the domain specified by the domain data file.

2.3 Namelist

On startup, the model reads an input namelist parameter file from stdin. The model has very few input namelist parameters. Typically the only input parameters that are required are those that specify how often the model will communicate with the coupler, and those that specify the input data files. Some can also alter the fluxes sent to the coupler. See the section on input namelist variables for a complete list and description of namelist parameters.

3 Namelist

```
The model reads an input namelist from stdin at runtime.
Following is a list and description of available namelist input parameters.

case_name
     Type: character(len=16)
     Default: " "
     Required: no, but highly recommended
     Description: This is the case name text string that appears in output files
     to help identify the model run.

case_desc
     Type: character(len=64)
     Default: " "
     Required: no, but highly recommended
     Description: This is a short text string which is included in output files
     to help identify the model run.

rest_type
     Type: character(len=16)
     Default: "continue"
     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.

rest_pfile
     Type: character(len=256)
     Default: "./rpointer"
     Required: no
     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.

rest_bfile
     Type: character(len=256)
     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:
     * "null:" indicates no retrieval -- associated data file has been
               prepositioned into the current working directory.
     * "cp:"   or no-prefix indicates a normal unix file copy to an NFS mounted
               file system.
     * "mss:"  indicates retrieval from NCAR's Mass Storage System (MSS)

rest_date
     Type: integer
     Default: 00010101 (year 1, 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).

ncpl
     Type: integer
     Default: 24
     Required: no
     Description: This specifies how many times per day the model communicates
     (exchanges data) with the coupler.

data_file
     Type: character(len=256)
     Default: "null"
     Required: yes
     Description: This specifies the name of the netCDF input file in which ice
     fraction data is found.  The directory in which this file is found is
     specified by the data_dir namelist variable.

data_dir
     Type: character(len=256)
     Default: "null:"
     Required: no
     Description:
     The specifies the directory in which the input ice fraction data file
     is found (see the data_file namelist variable).  Valid prefix options are:
     * "null:" indicates no archival -- associated data file has been
               prepositioned into the current working directory.
     * "cp:"   or no-prefix indicates a normal unix file copy to an NFS mounted
               file system.
     * "mss:"  indicates retrieval from NCAR's Mass Storage System (MSS)
     Eg. data_dir = "cp:/usr/jdoe/" requests a local copy from the directory
     /usr/jdoe.

domain_file
     Type: character(len=256)
     Default: "null"
     Required: yes
     Description: This specifies the name of the netCDF file in which input
     domain specification data is found.  The directory in which this file is
     found is specified by the domain_dir namelist variable.

domain_dir
     Type: character(len=256)
     Default: "null:"
     Required: no
     Description:
     The specifies the directory in which the input domain specification data
     file is found (see the domain_file namelist variable).  Note that a prefix
     like "null:" is used to indicate a file archival device.
     Valid prefix options are:
     * "null:" indicates no retrieval -- associated data file has been
               prepositioned into the current working directory.
     * "cp:"   or no-prefix indicates a normal unix file copy from an NFS
       mounted file system.
     * "mss:"  indicates retrieval from NCAR's Mass Storage System (MSS)
     Eg. data_dir = "cp:/usr/jdoe/" requests a local copy from the directory
     /usr/jdoe.

flux_acc
     Type: integer boolean
     Default: 0 (false/off)
     Required: no
     Description: If activated, the dice model accumulates ice formed in the
     ocean component -- this a flux field sent from the ocean to the ice.  This
     accumulated ice can subsequently be melted back into the ocean component
     if and when there is sufficient melting potential.  Melting potential is
     also a field sent from the ocean to the ice component.

flux_swpf
     Type: real
     Default: 0.0
     Required: no
     Description: This is the "short-wave penetrating factor".  This is the
     fraction of net solar absorbed by the ice model the penetrates through
     the ice and ends up being absorbed into the ocean below.

mss_dir
     Type: character(len=256)
     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 Mass Storage System (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:").

mss_rtpd
```

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

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

info_dbug
     Type: integer
     Default: 1
     Required: no
     Description: Debugging information level: 0, 1, 2, or 3.
     * 0 => write the least amount of debugging information to stdout
     * 1 => write a  small  amount of debugging information to stdout
     * 2 => write a  medium amount of debugging information to stdout
     * 3 => write a  large  amount of debugging information to stdout
```

4 Output Datasets

4.1 History Files

The data ice model does not create history files. The only data associated with this model is the data that is already contained in the input datasets.

4.2 Restart Files

The dice model creates restart files. The only data on the restart files is *accumulated water*, which is ice formed by the ocean component that is stored in dice so that it can be subsequently melted back into the ocean component (thus conserving salinity). By default the dice model does not accumulate ice formed by the ocean, and thus in default mode the dice model does not actually need the data on its restart file, although it always creates restart files anyway. The ability to accumulate water is activated by the namelist parameter *flux_ Qacc*. See the Namelist section of this document.

4.3 Runtime Diagnostics

The data ice model generates diagnostic messages which are written to stdout. This 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 model input parameters.

5 Data Exchanged with the Coupler

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.

5.1 Units Convention

All data exchanged conforms to this units convention:

```
Sign convention:
     positive value <=> downward flux

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
```

5.2 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).

5.2.1 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.
```

5.2.2 Data Received from Coupler

```
  time coordination data
* date, seconds: the exact time the coupler will start the simulation from.
```

5.3 Time Variant Data

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

Each data 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 data models must insure that fluxes sent to the coupler are appropriate in this context.

5.3.1 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)
```

5.3.2 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)
```

5.3.3 How Output Data is Derived

Ice extent (fractional coverage at each grid cell) is obtained by reading in climatological monthly mean ice fraction data from an input data file and linear interpolation in time.

The ice surface temperature is computed according to a formula that roughly approximates the ice temperature found in standalone versions of CCM2 (an older version of the CCSM atmosphere component). The formula is (for the northern hemisphere):

$$T_{ice} = 260 + 10cos(2\pi(rmonth - 8)/12)$$

where rmonth = 0 corresponds to January 1st, and rmonth = 6 corresponds to July 1st.

Surface albedos sent to the coupler are somewhat arbitrary but are based on reasonable values for snow and ice albedos combined with an assumption about the fraction of surface snow vs. ice. The albedo values are:

$$\begin{aligned} \alpha_{visible,direct} &= 0.586 \\ \alpha_{near-infrared,direct} &= 0.774 \\ \alpha_{visible,diffuse} &= 0.557 \\ \alpha_{near-infrared,diffuse} &= 0.772 \end{aligned}$$

The atmosphere/ice sensible, latent, and upward-longwave heat fluxes, evaporation, surface stress, and 2-meter reference temperatures are calculated according to an involved formula that is documented in the CCSM Coupler (cpl) document. See that document for details. The Coupler does a similar calculation for atmosphere/ocean fluxes.

The default value of penetrating short-wave radiation is zero. Optionally, by changing the namelist parameter flux_swpf ("short-wave penetrating factor"), will allow an arbitrary fraction of net solar absorbed to pass through the ice component and be absorbed into the ocean below.

Melt water is set to zero by default. Optionally, the flux_acc namelist parameter activates the dice model's ability to accumulate ice formed in the ocean component. This ice formed is flux field sent from the ocean to the ice. This accumulated ice can subsequently be melted back into the ocean component if and when there is sufficient melting potential. Melting potential is also a field sent from the ocean to the ice component.

Salt flux is alway set to zero.

The ice/ocean surface stress is set equal to the atmosphere/ice surface stress (see above).

6 Running the Data Model

6.1 Overview

The dice 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, as well as a coupler component. The scripts that build and run the CCSM system are described in detail in the CCSM 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 data model "setup scripts". A brief description of the CCSM run scripts is below. See the CCSM User's Guide for complete information.

6.2 Master Run Script and Component Setup Scripts

Two levels of c-shell scripts are used to build and run the CCSM system. 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 dice). Each CCSM component setup script is run in its own, separate subdirectory, where its executable resides and in which all of its input and output files are kept. 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. dice.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 climatological data files.

c. Build the component model 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.

7 Source Code Maintenance

7.1 Obtaining Source Code

The source code is available as part of the CCSM 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.

7.2 Data Model Source Code

The source code is written using standard Fortran 90.

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

7.3 Shared Source Code

The dice model source code itself (found in .../models/ice/dice/ in the CCSM distribution) 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 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 dependent 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.

7.4 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 dice. 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.