

**The CCSM
Climatological Data Land 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.cesm.ucar.edu/models>

CVS tag \$Name: cesm2_2_beta08 \$ Build date: June 17, 2004

Contents

1	Introduction	2
2	Input Datasets	2
2.1	Land Data	2
2.2	Domain Data	2
2.3	Namelist	2
3	Namelist	2
4	Output Datasets	5
4.1	History Files	5
4.2	Restart Files	5
4.3	Runtime Diagnostics	5
5	Data Exchanged with the Coupler	5
5.1	Units Convention	5
5.2	Time Invariant Data	6
5.2.1	Data Sent to Coupler	6
5.2.2	Data Received from Coupler	6
5.3	Time Variant Data	6
5.3.1	Data Sent to Coupler	7
5.3.2	Data Received from Coupler	7
5.3.3	How Output Fields are Derived	8
6	Running the Data Model	8
6.1	Overview	8
6.2	Master Run Script and Component Setup Scripts	8
7	Source Code Maintenance	9
7.1	Obtaining Source Code	9
7.2	Data Model Source Code	9
7.3	Shared Source Code	9
7.4	Shared Build Environment	9

1 Introduction

The Climatological Data Land Model (dlnd) functions as the land component in a CCSM configuration. Recall that a *configuration* consists of various independent *component models* (atmosphere, sea-ice, land, and ocean), each connected to a *coupler*. The dlnd land component interacts with the coupler just like any land model would, but it is not an active model, rather, it takes land data from an input data file and sends it to the coupler, ignoring any forcing data received from the coupler. Typically the input data file contains mean monthly data generated by an active CCSM land model (it's unlikely that real world observations exist for all the required fields). Such a "dummy" land model is useful for doing ocean + ice spinup runs.

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

2.1 Land Data

The model cycles thru netCDF data files containing all the data the coupler expects from a land model. Each file contains one year's worth of data consisting of 12 sets of mean monthly fields. This input data must have the same domain (resolution and coordinate arrays) as the land model. The model can cycle thru a multi-year sequence of data. Generally the input data files are created from history files that are output from the CCSM coupler component (cpl) using data provided by an active CCSM land component (*e.g.* CLM2). These files could, hypothetically, contain data that originates from any source, as long as the file format remains the same. Because netCDF files are self describing, one can query the file itself for specifics about the file format.

2.2 Domain Data

On startup, the 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, grid cell areas, and a domain mask. The model uses a rectilinear grid with 2d coordinate arrays x(i,j) & y(i,j). Because the input land data (above) is on the same domain as the dlnd model, and because the input land data contains domain information, any input land data file can also serve as a domain data file. Normally the first file in the input data file sequence is used as 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 sequence of input land data files. 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=256)
Default: " "
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.

ncpl

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

data_format

Type: character(len=64)
Default: 'null'
Required: no
Description: Generally not used. If data_format = "dlnd5.0" then the model expects input data to be in the same format as was used in version 5.0 and previous versions. Note: the 5.0 version was part of the ccsm2.0 package. This backward compatibility is a depreciated feature that is unlikely to be supported in future releases.

data_dir

Type: character(len=256)
Default: 'null'
Required: yes
Description: This specifies where the land data input file sequence is found. Examples:
(1) data_dir = "cp:~jdoe/data " => look for input data in user djoe's home directory in the data subdirectory.
(2) data_dir = "mss:/JDOE/data " => look for input data on NCAR's Mass Storage System archival device.

data_fname

Type: character(len=256)
Default: 'null'
Required: yes
Description: This specifies file names of the input data files. The data files are assumed to contain 12 months worth of mean monthly data. The file name can be any valid unix file name. This is a template file name, that is, somewhere in the file name must be the text substring "yyyy" which will be replaced by a specific year to create the actual file name. For example, if data_fname = "b30.004.cpl6.ha.yyyy-xx.040520.nc" and the dlnd model needs data for year 750, it will construct the file name

"b30.004.cpl6.ha.0750-xx.040520nc". This file must be located in the data_dir described above.

data_year0

Type: integer

Default: 1

Required: no

Description: This specifies the first year in the input data file sequence.

data_nyear

Type: integer

Default: 1

Required: no

Description: This specifies the number of years in the input data file sequence.

data_oyear

Type: integer

Default: 1

Required: no

Description: This specifies the "offset year" for the input data file sequence. The data file for data_year0 will coincide with the simulation year data_oyear.

data_aroff

Type: character(len=256)

Default: "null"

Required: no

Description: This can be used to specify an "auxilliary" (non-standard) runoff data file. This option is for power-users only.

By default, dlnd5 reads runoff data from the same data files it uses for all other data. In this case the runoff data is on a 1/2 degree grid that is the standard runoff grid for CCSM2.0. It is possible, however, to replace this standard runoff data with "auxiliary" runoff data from a separate data file. This alternate data can even be on a non-standard grid (for example the 19-basins-of-runoff grid used in CCSM1.x codes).

* data_aroff = "null" => the runoff data sent to the coupler is on the standard 1/2 degree runoff grid used in CCSM2.0

* data_aroff = "zero" => the runoff data sent to the coupler is on the standard 1/2 degree runoff grid used in CCSM2.0, but the runoff data is set to 0.0 everywhere. This option doesn't require a new runoff data file, it merely sends all-zero runoff data to the coupler.

* data_aroff = "UserCreatedFile" => the runoff data sent to the coupler is found in a user created data file (an "auxiliary" data file). See the source code file called "runoff_aux.F90" for details of how this works.

info_debug

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 land 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 data land model does not need or create restart files.

4.3 Runtime Diagnostics

The data land 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 \Leftrightarrow 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
* 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)
```

5.3.2 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)
```


5.3.3 How Output Fields are Derived

Data from the input data sequence is assumed to be monthly average data. This data is linearly interpolated in time to get instantaneous fields, and this data is sent to the coupler. All data sent to the coupler is taken directly from the input data sequence. For example, instantaneous albedos are derived by interpolating between two monthly averaged albedo fields.

6 Running the Data Model

6.1 Overview

The dlnd 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 dlnd). 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. dlnd.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.cesm.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 dlnd model source code itself (found in `.../models/lnd/dlnd6/` 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 dlnd. 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.