The CCSM Observational Data Atmosphere Model (latm6)

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

CCSM coupled model is based on a framework which divides the complete climate system into *component models* connected by a *coupler*. This design requires four component models – atmosphere, sea-ice, land, and ocean – each connected to the Coupler, and each exchanging data with the Coupler only.

The Observational Data Atmosphere Model (latm) functions as the atmosphere component in a CCSM configuration. The latm atmosphere component interacts with the Coupler just like any atmosphere model would, but it is not an active model, rather, it takes atmosphere data from input data files and sends it to the Coupler, ignoring any forcing data received from the Coupler. Typically the input data files contain climatological or time averaged observational data, although some data fabrication is usually necessary as it's unlikely that real world observations exists for all the required fields. Such a "dummy" atmosphere model is useful for doing ocean + ice spinup runs.

This *latm* code is a variant of the *datm* code, the difference being that the datm cycles thru daily average data fields normally created by the CCSM active atmosphere component (CCM, now called CAM), while the latm cycles thru data from other sources (NCEP in particular). The latm cycles thru separate data file streams for atmospheric states, precipitation, and radiation. Further, these three data streams can have different sampling intervals, for example, radiation data can be daily average data while precipitation data is monthly average data. *Note that this version of latm can only use NCEP data for all three data streams and the biases in these data produce solutions that are not of high scientific quality. The latm code is provided as an example of how one group of users altered the datm source code to cycle through different types of input data streams.*

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 Atmospheric Data

The latm will cycle through a given set of state, precipitation, and radiation data sets. Each of these three data streams must have the same spatial resolution, e.g. T62, but the can differ in their temporal resolutions. For example, the state and precipitation data sets can be 6-hourly and monthly data, respectively. In its present configuration, the latm uses NCEP reanalysis data sets for all three forcing streams. The state variables consist of zonal and meridional 10m winds, surface 2m (extrapolated to 10m) air temperature, specific humidity, and density. The radiation data sets are long-wave down, short-wave down, and short-wave up.

All of the forcing information is set-up using four namelist variables per forcing type:

1. data_dir[1...3]: a directory path for the forcing data files. It must include the name of the data source in capital letters, e.g. "NCEP."

2. data_year0[1...3]: the 1st year in the data sequence, e.g. 1985.

3. data_oyear[1...3]: the simulation year matched to year0, e.g. 1.

4. data_nyear[1...3]: the number of years in the sequence, e.g. 2.

So, in the above example, the NCEP reanalysis fields for 1985 (corresponding to simulation year 1) and 1986 will be cycled through continuously. The array indices of the these namelist variables correspond to state, precipitation, and radiation data streams, respectively.

Note that this version of latm can only use NCEP data for all three data streams and the biases in these data produce solutions that are not of high scientific quality. The latm code is provided as an example of how one group of users altered the datm source code to cycle through different types of input data streams.

2.2 Domain Data

On startup, the model reads in domain data from the forcing data files. Data exchanged with the coupler will be on this model domain. This file contains x & y coordinate arrays. The model uses a rectilinear grid with 2d coordinate arrays x(i,j) & y(i,j).

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 atmosphere 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: "null"
     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: "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.

data_type[1...3]
     Type: character(len=16)
     Default: "fabricate"
     Required: no
     Description: This specifies the type of data to be used for state,
     precipitation, and radiation fields, respectively.  Examples:
     (1) data_type = "fabricate" => make domain data and use constant fields;
         no data is read in.  NOTE:  If any or all of the data_type elements are
         set to "fabricate", all fields are fabricated.
     (2) data_type = "NCEP" => read in NCEP data.
     (3) data_type = "CCM" => read in data from an active atmosphere simulation.

data_dir[1...3]
     Type: character(len=256)
     Default: "null:"
     Required: yes
     Description: This specifies where the data input file sequences
     are found for state, precipitation, and radiation data files, respectively.
     Examples:
     (1) data_dir = "null:/whatever " => the "null:" prefix indicates that the
         data has already been pre-positioned in the exectution directory.
     (2) data_dir = "mss:/JDOE/data " => look for input data on NCAR's
          Mass Storage System archival device.
     (3) data_dir = "cp:~jdoe/data " => look for input data in user jdoe's
         home directory in the data subdirectory.

data_year0[1...3]
     Type: integer
     Default: 1
     Required: yes (unless the data year happens to be 1)
     Description: the 1st year in the state, precipitation, and radiation
     data files, respectively, e.g. 1985.

data_oyear[1...3]
     Type: integer
     Default: 1
     Required: yes
     Description: the simulation year matched to year0 for the state,
     precipitation, and radiation data files, respectively, e.g. 1.

data_nyear[1...3]
     Type: integer
     Default: 1
     Required: no (unless you want to cycle thru more than one year)
     Description: the number of years in the sequence for the state,
     precipitation, and radiation files, respectively, e.g. 2.

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.

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


flux_albfb
     Type: integer boolean (0 <=> false)
     Default: 0
     Required: no
     Description: If flux_albfb = true, then albedo feedback is activated.
```

```
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 atmosphere 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 atmosphere model does not need or create restart files.

4.3 Runtime Diagnostics

The data atmosphere 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 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)
```

5.3.2 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)
```

5.3.3 How Output Fields are Derived

Data from the input data sequence is assumed to be daily 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, with these exceptions:

1. If four components of shortwave down are not present on the input data files, they are derived from the net downward shortwave (which must be in the input data files):
   - (shortwave down, visible/direct) = 0.28 * (shortwave down, net)
   - (shortwave down, visible/diffuse) = 0.24 * (shortwave down, net)
   - (shortwave down, near-infrared/direct) = 0.31 * (shortwave down, net)
   - (shortwave down, near-infrared/diffuse) = 0.17 * (shortwave down, net)

   $$shortwave_{down,net} = \sum_{m=1}^{4} shortwave_{down}^{m}$$

2. Optionally (by explicitly selecting a namelist option), all shortwave fields (net and downward) can be multiplied by an arbitrary factor. See the flux_swfact option in the Namelist section of this document.

3. If the user explicitly activates the albedo feedback option (by default albedo feedback is not activated), the net shortwave radiation sent to the coupler is derived using the four downward shortwave components plus the albedos received from the coupler:

   $$shortwave_{net} = \sum_{m=1}^{4} (1 - albedo^{m}) * shortwave_{down}^{m}$$

   This allows the net shortwave radiation to be more consistent with the surface albedos (e.g. sea ice extent). See the flux_albfb option in the Namelist section of this document.

   Note that this means that the latm does not ignore all input from the coupler – latm output to the coupler is now a function of input from the coupler.

4. If sea-level pressure is missing from the input data files, then sea-level pressure is set equal to the bottom layer pressure. If both sea-level and bottom layer pressure is missing from the input data files, both fields are set to 1.0e30.

5. It is assumed that precipitation data on the input data files is not divided into snow vs. rain. Where ever the bottom level atmosphere temperature is below freezing, latm interprets precipitation to be snow. Where ever the bottom level atmosphere temperature is above freezing, latm interprets precipitation to be rain.

6 Running the Data Model

6.1 Overview

The latm 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 latm). 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. latm.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 latm model source code itself (found in .../models/atm/latm/ 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 latm. 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.