User’s Guide to the Community Atmosphere Model CAM-4.0

Brian Eaton
NCAR

User’s Guide to the Community Atmosphere Model CAM-4.0

by Brian Eaton

Table of Contents

Acknowledgments

The authors wish to acknowledge members of NCAR's Atmospheric Modeling and Predictability Section (AMP), CCSM Software Engineering Group (CSEG), and Computation and Information Systems Laboratory (CISL) for their contributions to the development of CAM-4.0.

The new model would not exist without the significant input from members of the CCSM Atmospheric Model Working Group (AMWG) too numerous to mention. Leo Donner (GFDL), Minghua Zhang (SUNY) and Phil Rasch (PNNL) were the co-chairs of the AMWG during the development of CAM-4.0.

We would like to acknowledge the substantial contributions to the CAM-4.0 effort from the National Science Foundation, Department of Energy, the National Oceanic and Atmospheric Administration, and the National Aeronautics and Space Administration.

Chapter 1. Introduction

Note: This User's Guide is under active development. It was last modified on 3 April 2010.

Version 4.0 of the Community Atmosphere Model (CAM) is the latest in a series of global atmosphere models originally developed at the National Center for Atmospheric Research (NCAR). The current development of CAM is guided by the Atmosphere Model Working Group[1] (AMWG) of the Community Climate System Model[2] (CCSM) project. CAM is used as both a standalone model and as the atmospheric component of the CCSM. CAM has a long history of use as a standalone model by which we mean that the atmosphere is coupled to an active land model (CLM), a thermodynamic only sea ice model (CICE), and a data ocean model (DOCN). When one speaks of "doing CAM simulations" the implication is that it's the standalone configuration that is being used. When CAM is coupled to active ocean and sea ice models then we refer to the model as CCSM.

In versions of CAM before 4.0 the driver for the standalone configuration was completely separate code from what was used to couple the components of CCSM. One of the most significant software changes in CAM-4.0 is a refactoring of how the land, ocean, and sea ice components are called which enabled the use of the CCSM coupler to act as the CAM standalone driver (note that this also depended on the complete rewritting of the CCSM coupler to support sequential execution of the components). It is now accurate to say that the CAM standalone configuration is nothing more than a special configuration of CCSM in which the active ocean and sea ice components are replaced by data ocean and thermodynamic sea ice components.

Since the CAM standalone model is just a special configuration of CCSM it can be run using the CCSM scripts. This is done by using one of the "F" compsets and is described in the CCSM-4.0 User's Guide[3]. The main advantage of running CAM via the CCSM scripts is to leverage the high level of support that those scripts provide for doing production runs of predefined experiments on supported platforms. The CCSM scripts do things like: setting up reasonable runtime environments; automatically retrieving required input datasets from an SVN server; and archiving output files. But CAM is used in alot of environments where the complexity of production ready scripts is not necessary. In these instances the flexibility and simplicity of being able to completely describe a run using a short shell script is a valuable option. In either case though, the ability to customize a CAM build or runtime configuration depends on being able to use the utilities described in this document. Any build configuration can be set up via appropriate commandline arguments to CAM's **configure** utility, and any runtime configuration can be set up with appropriate arguments to CAM's **build-namelist** utility.

```
Much of the example code in this document is set off in sections like this.
Many examples refer to files in the distribution source tree using
filepaths that are relative to distribution root directory, which we denote
by $CAM_ROOT.  The notation indicates that CAM_ROOT is a shell variable
that contains the filepath.  This could just as accurately be referred to
as $CCSM_ROOT since the root directory of the CCSM distribution is the
same as the root of the CAM distribution which is contained within it.
```

Getting Help -- Other User Resources

The CAM Web Page

The central source for information on CAM is the CAM web page[4].

The CAM Bulletin Board

The CAM Bulletin Board is a moderated forum for rapid exchange of information, ideas, and topics of interest relating to the various versions of CAM. This includes sharing software tools, datasets, programming tips and examples, as well as discussions of questions, problems and workarounds. The primary motivation for the establishment of this forum is to facilitate and encourage communication between the users of the CAM around the world. This bulletin board will also be used to distribute announcements related to CAM.

The CAM Bulletin Board is here: http://bb.cgd.ucar.edu/.

Reporting bugs

If a user should encounter bugs in the code (i.e., it doesn't behave in a way in which the documentation says it should), the problem should be reported electronically to the CAM Bulletin Board[5]. When writing a bug report the guiding principle should be to provide enough information so that the bug can be reproduced. The following list suggests the minimal information that should be contained in the report:

1. The version number of CAM (or CCSM if CAM was obtained as part of a CCSM distribution).
2. The architecture on which the code was built. Include relevent information such as the Fortran compiler, MPI library, etc.
3. The configure commandline.
4. The build-namelist commandline.
5. Model printout. Ideally this would contain a stack trace. But it should at least contain any error messages printed to the output log.

Please note that that CAM is a research model, and not all features are supported.

Notes

1. http://www.ccsm.ucar.edu/working_groups/Atmosphere/
2. http://www.ccsm.ucar.edu/models/ccsm4.0/
3. http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/book1.html
4. http://www.ccsm.ucar.edu/models/ccsm4.0/cam
5. http://bb.cgd.ucar.edu/

Chapter 2. Building and Running CAM

This chapter describes how to build and run CAM in its standalone configuration. We do not provide scripts that are setup to work out of the box on a particular set of platforms. If you would like this level of support then consider running CAM from the CCSM scripts (see CCSM-4.0 User's Guide[1]). We do however provide some examples of simple run scripts which should provide a useful starting point for writing your own scripts (see the Section called *Sample Run Scripts*).

In order to build and run CAM the following are required:

- The source tree. Unlike previous CAM versions which were distributed independently of the CCSM, CAM4 will only be distributed with CCSM4. To obtain the source code go to the section "Acquiring the Code" on the CCSM Home Page[2]. When we refer to the root of the CAM source tree, this is the same directory as the root of the CCSM source tree. This directory is refered to throughout this document as $CAM_ROOT.
- Perl (version 5.4 or later).
- A GNU version of the **make** utility.
- Fortran90 and C compilers.
- A NetCDF library (version 3.6 or later) that has the Fortran APIs built using the same Fortran90 compiler that is used to build the rest of the CAM code. This library is used extensively by CAM both to read input datasets and to write the output datasets. The NetCDF source code is available here[3].
- Input datasets. The required datasets depend on the CAM configuration. Determining which datasets are required for any configuration is discussed in the Section called *Building the Namelist*. Acquiring those datasets is discussed in the Section called *Acquiring Input Datasets*.

To build CAM for SPMD execution it will also be necessary to have an MPI library (version 1 or later). As with the NetCDF library, the Fortran API should be build using the same Fortran90 compiler that is used to build the rest of CAM. Otherwise linking to the library may encounter difficulties, usually due to inconsistencies in Fortran name mangling.

Building and running CAM takes place in the following steps:

1. Configure model
2. Build model
3. Build namelist
4. Execute model

Configure model

This step is accomplished by running the **configure** utility to set the compile-time parameters such as the dynamical core (Eulerian Spectral, Semi-Lagrangian Spectral, Finite-Volume, or HOMME), horizontal grid resolution, and the type of parallelism to employ (shared-memory and/or distributed memory). The **configure** utility is discussed in Appendix A.

Build model

This step includes compiling and linking the executable using the GNU make command (**gmake**). **configure** creates a Makefile in the directory where the build is to take place. The user then need only change to this directory and execute the **gmake** command.

Build namelist

This step is accomplished by running the **build-namelist** utility, which supports a variety of options to control the run-time behavior of the model. Any namelist variable recognized by CAM can be changed by the user via the **build-namelist** interface. There is also a high level "use case" functionality which makes it easy for the user to specify a consistent set of namelist variable settings for running particular types of experiments. The **build-namelist** utility is discussed in Appendix B.

Execute model

This step includes the actual invocation of the executable. When running using distributed memory parallelism this step requires knowledge of how your machine invokes MPI executables. When using shared-memory parallelism using OpenMP you may also set the number of OpenMP threads. On most HPC platforms access to the compute resource is through a batch queue system. The sample run scripts discussed in the Section called *Sample Run Scripts* show how to set the batch queue resources on several HPC platforms.

Sample Interactive Session

The following sections present an interactive C shell session to build and run a default version of CAM. Most often these steps will be encapsulated in shell scripts. An important advantage of using a script is that it acts to document the run you've done. Knowing the source code tree, and the **configure** and **build-namelist** commands provides all the information needed to exactly replicate a run.

For the interactive session the shell variable `camcfg` is set to the directory in the source tree that contains the CAM **configure** and **build-namelist** utilities ($CAM_ROOT/models/atm/cam/bld).

Configuring CAM for serial execution

We start by changing into the directory in which the CAM executable will be built, and then setting the environment variables INC_NETCDF and LIB_NETCDF which specify the locations of the NetCDF include files and library. This information is required by **configure** in order for it to produce the `Makefile`. The NetCDF library is require by all CAM builds. The directories given are just examples; the locations of the NetCDF include files and library are system dependent. The information provided by these environment variables could alternatively be provided via the commandline arguments **`-nc_inc`** and **`-nc_lib`**.

```
% cd /work/user/cam_test/bld
% setenv INC_NETCDF /usr/local/include
% setenv LIB_NETCDF /usr/local/lib
```

Next we issue the **configure** command. The argument **`-dyn fv`** specifies using the FV dynamical core which is the default for CAM4, but we recommend always adding the dynamical core (aka dycore) argument to **configure** commands for clarity. The argument **`-hgrid 10x15`** specifies the horizontal grid. This is the coarsest grid available for the FV dycore in CAM and is often useful for testing purposes.

We recommend using the **`-test`** option the first time CAM is built on any machine. This will check that the environment is properly set up so that the Fortran compiler works and can successfully link to the NetCDF and MPI (if SPMD is enabled) libraries. Furthermore, if the configuration is for serial execution, then the tests will include both build and run phases which may be useful in exposing run time problems that don't show up during the build, for example when libraries are linked dynamically. If any tests fail then it is useful to rerun the **configure** command and add the **`-v`** option which will produce verbose output of all aspects of the configuration process including the tests. If the configuration is for an SPMD build, then no attempt to run the tests will be made. Typically MPI runs must be submitted to a batch queue and are not enabled from interactive sessions. But the build and static linking will still be tested.

```
% $camcfg/configure -dyn fv -hgrid 10x15 -nospmd -nosmp -test
Issuing command to the CICE configure utility:
  $CAM_ROOT/models/ice/cice/bld/configure -hgrid 10x15 -cice_mode prescribed \
  -ntr_aero 0 -ntr_pond 1 -ntr_iage 0 -ntasks 1 -nthreads 1 \
  -cache config_cache_cice.xml -cachedir /work/user/cam_test
configure done.
creating /work/user/cam_test/bld/Filepath
creating /work/user/cam_test/bld/misc.h
creating /work/user/cam_test/bld/preproc.h
creating /work/user/cam_test/bld/Makefile
creating /work/user/cam_test/bld/config_cache.xml
Looking for a valid GNU make... using gmake
Test linking to NetCDF library... ok
configure done.
```

The first line of output from the **configure** command is an echo of the system command that CAM's **configure** issues to invoke the CICE **configure** utility. This brings up a major difference from the CAM3 build. The thermodynamic sea ice model used by CAM3 was a version of CSIM4 that was modified to run using CAM's decomposition and data structures. It was essentially a part of CAM and was built just as any other physics parameterization. The thermodynamic sea ice model used by CAM4 is a special configuration of CICE4 which is a completely independent component with it's own build requirements. A major build requirement of the CICE model is that it's grid decomposition (which is independent of CAM's decomposition even when the two models are using the same horizontal grid) must be specified at build time. The CICE **configure** utility is responsible for setting the values of the CPP macros that are needed to build the CICE code. These settings include the specification of the CICE decomposition. Note that the line "configure done." immediately after the CICE **configure** commandline is being issued by the CICE **configure**, not by CAM's **configure**.

The next five lines of output inform the user of the files being created by **configure**. All the files produced by **configure** except for the cache file are required to be in the CAM build directory, so it is generally easiest to be in that directory when **configure** is invoked. Note: the files `misc.h` and `preproc.h` are no longer used by CAM, but are required for the CLM build.

The output from the **`-test`** option tells us that **gmake** is a GNU Make on this machine, and that the Fortran compiler can successfully link to the NetCDF library. Since this is a serial configuration no test for linking to the MPI library was done.

Configuring CAM for parallel execution

Before moving on to building CAM we address configuring the executable for parallel execution. But before talking about configuration specifics let's briefly discuss the parallel execution capabilities of CAM.

CAM makes use of both distributed memory parallelism implemented using MPI (referred to throughout this document as SPMD[4]), and shared memory parallelism implemented using OpenMP (referred to as SMP). Each of these parallel modes may be used independently of the other, or they may be used at the same time which we refer to as "hybrid mode". When talking about the SPMD mode we usually refer to the MPI processes as "tasks", and when talking about the SMP mode we usually refer to the OpenMP processes as "threads". A feature of CAM which is very helpful in code development work is that the simulation results are independent of the number of tasks and threads being used.

Now consider configuring CAM to run in pure SPMD mode. With CAM3 SPMD was turned on using the **-spmd** option. But with CAM4 if we try that we find the following:

```
% $camcfg/configure -dyn fv -hgrid 10x15 -spmd -nosmp
**    ERROR: If CICE decomposition parameters are not specified, then
**    -ntasks must be specified to determine a default decomposition
**    for a pure MPI run.  The setting was:  ntasks=
```

This error results from the fact discussed in the Section called *Configuring CAM for serial execution* that the CICE model needs to set it's decomposition at build time, and in order to set the decomposition it needs to know how much parallelism is going to be used. If you know how the CICE decomposition works then you're free to set it explicitly using the **configure** options provided for that purpose. Otherwise it's best to let the CICE **configure** set the decomposition for you and just specify the number of MPI tasks that the job will use via setting the **-ntasks** option as follows:

```
% $camcfg/configure -dyn fv -hgrid 10x15 -ntasks 6 -nosmp
Issuing command to the CICE configure utility:
  $CAM_ROOT/models/ice/cice/bld/configure -hgrid 10x15 -cice_mode prescribed \
  -ntr_aero 0 -ntr_pond 1 -ntr_iage 0 -ntasks 6 -nthreads 1 \
  -cache config_cache_cice.xml -cachedir /work/user/cam_test
configure done.
...
```

Notice that the number of tasks specified to CAM's **configure** is passed through to the commandline that invokes the CICE **configure**. Generally any number of tasks that is appropriate for CAM to use for a particular horizontal grid will also work for CICE. But it is possible to get an error from CICE at this point in which case either the number of tasks requested should be adjusted, or the options that set the CICE decomposition explicitly will need to be used.

Note: The use of the **-ntasks** argument to **configure** implies building for SPMD. This means that an MPI library will be required. Hence, the specification **-ntasks 1** is not the same as building for serial execution which is done via the **-nospmd** option and does not require a full MPI library. (Implementation detail: when building for serial mode a special serial MPI library is used which basically provides a complete MPI API, but doesn't do any message passing.)

Next consider configuring CAM to run in pure SMP mode. With CAM3 SMP was turned on using the **-smp** option. But with CAM4 that will result in the same error from CICE that we obtained above from attempting to use **-spmd**. If we are going to run the CICE code in parallel, we need to specify up front how much parallelism will be used so that the CICE **configure** utility can set the CPP macros that determine the grid decomposition. We specify the amount of SMP parallelism by setting the **-nthreads** option as follows:

```
% $camcfg/configure -dyn fv -hgrid 10x15 -nospmd -nthreads 6
Issuing command to the CICE configure utility:
  $CAM_ROOT/models/ice/cice/bld/configure -hgrid 10x15 -cice_mode prescribed \
  -ntr_aero 0 -ntr_pond 1 -ntr_iage 0 -ntasks 1 -nthreads 6 \
  -cache config_cache_cice.xml -cachedir /work/user/cam_test
configure done.
...
```

We see that the number of threads has been passed through to the CICE **configure** command.

Note: The use of the **-nthreads** argument to **configure** implies building for SMP. This means that the OpenMP directives will be compiled. Hence, the specification **-nthreads**

> `1` is not the same as building for serial execution which is done via the **`-nosmp`** option and does not require a compiler that supports OpenMP.

Finally, to configure CAM for hybrid mode, simply specify both the **`-ntasks`** and **`-nthreads`** arguments to **configure**.

Building CAM

Once **configure** is successful, build CAM by issuing the make command:

```
% gmake -j2  >&! make.out
```

The argument **`-j2`** is given to allow a parallel build using 2 processes. The optimal number of processes to use depends on the compute resource available.

It is useful to redirect the output from **make** to a file for later reference. This file contains the exact commands that were issued to compile each file and the final command which links everything into an executable file. Relevant information from this file should be included when posting a bug report concerning a build failure.

Building the Namelist

The first step in the run procedure is to generate the namelist files. The only safe way to generate consistent namelist settings is via the **build-namelist** utility. Even in the case where only a slight modification to the namelist is desired, the best practice is to provide the modified value as an argument to **build-namelist** and allow it to actually generate the namelist files.

The following interactive C shell session builds a default namelist for CAM. We assume that a successful execution of **configure** was performed in the build directory as discussed in the previous section. This is an essential prerequisite because the `config_cache.xml` file produced by **configure** is a required input file to **build-namelist**. One of the responsibilities of **build-namelist** is to set appropriate default values for many namelist variables, and it can only do this if it knows how the CAM executable was configured. That information is present in the cache file. As in the previous section the shell variable `camcfg` is set to the CAM configuration directory ($CAM_ROOT/models/atm/cam/bld).

We begin by changing into the directory where CAM will be run. It is usually convenient to have the run directory be separate from the build directory. Possibly a number of different runs will be done that each need to have a separate run directory for the output files, but will all use the same executable file from a common build directory.

Next we set the CSMDATA environment variable to point to the root directory of the tree containing the input data files. Note that this is a required input for **build-namelist** (this information may alternatively be provided using the **`-csmdata`** argument). If not provided then **build-namelist** will fail with an informative message. The information is required because many of the namelist variables have values that are absolute filepaths. These filepaths are resolved by **build-namelist** by prepending the CSMDATA root to the relative filepaths that are stored in the default values database.

The **build-namelist** commandline contains the **`-config`** argument which is used to point to the cache file which was produced in the build directory. It also contains the **`-test`** argument, explained further below.

```
% cd /work/user/cam_test
% setenv CSMDATA /fs/cgd/csm/inputdata
% $camcfg/build-namelist -test -config /work/user/cam_test/bld/config_cache.xml
```

```
Writing CICE namelist to ./ice_in
Writing CLM namelist to ./lnd_in
Writing driver namelist to ./drv_in
Writing dry deposition namelist to ./drv_flds_in
Writing ocean component namelist to ./ocn_in
Writing CAM namelist to ./atm_in
Checking whether input datasets exist locally...
OK -- found prescribed_aero_file = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/
OK -- found prescribed_aero_datapath = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_a
OK -- found ncdata = /fs/cgd/csm/inputdata/atm/cam/inic/fv/cami_0000-01-01_10x15_L26_c0
OK -- found bnd_topo = /fs/cgd/csm/inputdata/atm/cam/topo/USGS-gtopo30_10x15_remap_c050
OK -- found absems_data = /fs/cgd/csm/inputdata/atm/cam/rad/abs_ems_factors_fastvx.c030
OK -- found bndtvs = /fs/cgd/csm/inputdata/atm/cam/sst/sst_HadOIBl_bc_10x15_clim_c05052
OK -- found focndomain = /fs/cgd/csm/inputdata/atm/cam/ocnfrac/domain.camocn.10x15_USGS
OK -- found bndtvs = /fs/cgd/csm/inputdata/atm/cam/sst/sst_HadOIBl_bc_10x15_clim_c05052
OK -- found focndomain = /fs/cgd/csm/inputdata/atm/cam/ocnfrac/domain.camocn.10x15_USGS
OK -- found tropopause_climo_file = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart/ub/c
OK -- found faerdep = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/aero/aerosolc
OK -- found fpftcon = /fs/cgd/csm/inputdata/lnd/clm2/pftdata/pft-physiology.c100226
OK -- found fsnowaging = /fs/cgd/csm/inputdata/lnd/clm2/snicardata/snicar_drdt_bst_fit_
OK -- found fatmlndfrc = /fs/cgd/csm/inputdata/lnd/clm2/griddata/fracdata_10x15_USGS_07
OK -- found fsnowoptics = /fs/cgd/csm/inputdata/lnd/clm2/snicardata/snicar_optics_5bnd_
OK -- found fsurdat = /fs/cgd/csm/inputdata/lnd/clm2/surfdata/surfdata_10x15_simyr2000_
OK -- found fatmgrid = /fs/cgd/csm/inputdata/lnd/clm2/griddata/griddata_10x15_070212.nc
OK -- found prescribed_ozone_datapath = /fs/cgd/csm/inputdata/atm/cam/ozone
OK -- found prescribed_ozone_file = /fs/cgd/csm/inputdata/atm/cam/ozone/ozone_1.9x2.5_I
OK -- found rad_climate for D_sulf:/fs/cgd/csm/inputdata/atm/cam/physprops/sulfate_camr
OK -- found rad_climate for D_dust1:/fs/cgd/csm/inputdata/atm/cam/physprops/dust1_camrt
OK -- found rad_climate for D_dust2:/fs/cgd/csm/inputdata/atm/cam/physprops/dust2_camrt
OK -- found rad_climate for D_dust3:/fs/cgd/csm/inputdata/atm/cam/physprops/dust3_camrt
OK -- found rad_climate for D_dust4:/fs/cgd/csm/inputdata/atm/cam/physprops/dust4_camrt
OK -- found rad_climate for D_bcar1:/fs/cgd/csm/inputdata/atm/cam/physprops/bcpho_camrt
OK -- found rad_climate for D_bcar2:/fs/cgd/csm/inputdata/atm/cam/physprops/bcphi_camrt
OK -- found rad_climate for D_ocar1:/fs/cgd/csm/inputdata/atm/cam/physprops/ocpho_camrt
OK -- found rad_climate for D_ocar2:/fs/cgd/csm/inputdata/atm/cam/physprops/ocphi_camrt
OK -- found rad_climate for D_SSLTA:/fs/cgd/csm/inputdata/atm/cam/physprops/ssam_camrt_
OK -- found rad_climate for D_SSLTC:/fs/cgd/csm/inputdata/atm/cam/physprops/sscm_camrt_
```

The first six lines of output from **build-namelist** inform the user of the namelist files that have been created. There is a file for each of the physical model components (`ice_in`, `lnd_in`, `ocn_in`, `atm_in`), a file for the driver (`drv_in`), and a file that is read by both the atmosphere and land components (`drv_flds_in`). Note that these filenames are hardcoded in the components and my not be changed with source code modifications.

The next section of output is the result of using the **-test** argument to **build-namelist**. As with **configure** we recommend using this argument whenever a model configuration is being run for the first time. It checks that each of the files that are present in the generated namelists can be found in the input data tree whose root is given by the CSMDATA environment variable. If a file is not found then the user will need to take steps to make that file accessible to the executing model before a successful run will be possible. The following is a list of possible actions:

1. Acquire the missing file. If this is a default file supplied by the CCSM project then you will be able to download the file from the project svn data repository (see the Section called *Acquiring Input Datasets*).
2. If you have write permissions in the directory under $CSMDATA then add the missing file to the appropriate location there.
3. If you don't have write permissions under $CSMDATA then put the file in a place where you can (for example, your run directory) and rerun **build-namelist** with an explicit setting for the file using you specific filepath.

Expanding a bit on rerunning **build-namelist**: let's say for example that the **-test** option informed you that the **ncdata** file `cami_0000-01-01_1.9x2.5_L26_c070408.nc` was not found. You acquire the file from the data repository, but don't have permissions to write in the $CSMDATA tree. So you put the file in your run directory and issue a **build-namelist** command that looks like this:

```
% $camcfg/build-namelist -config /work/user/cam_test/bld/config_cache.xml \
  -namelist "&atm ncdata='/work/user/cam_test/cami_0000-01-01_1.9x2.5_L26_c070408.nc' /
```

Now the namelist in `atm_in` will contain an initial file (specified by namelist variable **ncdata**) which will be found by the executing CAM model.

Acquiring Input Datasets

> **Note:** If you are doing a standard production run that is supported in the CCSM scripts, then using those scripts will automatically invoke a utility to acquire needed input datasets. The information in this section is to aid developers using CAM standalone scripts.

The input datasets required to run CAM are available from a Subversion repository located here: https://svn-ccsm-inputdata.cgd.ucar.edu/trunk/inputdata/. The user name and password for the input data repository will be the same as for the code repository (which are provided to users when they register to acquire access to the CCSM source code repository).

Example

If you have a list of files that you need to acquire before running CAM, then you can either just issue commands interactively, or if your list is rather long then you may want to put the commands into a shell script. For example, suppose after running **build-namelist** with the **-test** option you find that you need to acquire the file `/fs/cgd/csm/inputdata/atm/cam/inic/fv/cami_0000-01-01_10x15_L26_c030918.nc`. And let's assume that `/fs/cgd/csm/inputdata/` is the root directory of the inputdata tree, and that you have permissions to write there. If the subdirectory `atm/cam/inic/fv/` doesn't already exist, then create it. Finally, issue the following commands at an interactive C shell prompt:

```
% set svnrepo='https://svn-ccsm-inputdata.cgd.ucar.edu/trunk/inputdata'
% cd /fs/cgd/csm/inputdata/atm/cam/inic/fv
% svn export $svnrepo/atm/cam/inic/fv/cami_0000-01-01_10x15_L26_c030918.nc
Error validating server certificate for 'https://svn-ccsm-inputdata.cgd.ucar.edu:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
 - The certificate hostname does not match.
 - The certificate has expired.
Certificate information:
 - Hostname: localhost.localdomain
 - Valid: from Feb 20 23:32:25 2008 GMT until Feb 19 23:32:25 2009 GMT
 - Issuer: SomeOrganizationalUnit, SomeOrganization, SomeCity, SomeState, --
 - Fingerprint: 86:01:bb:a4:4a:e8:4d:8b:e1:f1:01:dc:60:b9:96:22:67:a4:49:ff
(R)eject, accept (t)emporarily or accept (p)ermanently? p
A    cami_0000-01-01_10x15_L26_c030918.nc
Export complete.
```

The messages about validating the server certificate will only occur for the first file that you export if you answer "p" to the question as in the example above.

Running CAM

Once the namelist files have successfully been produced, and the necessary input datasets are available, the model is ready to run. Usually CAM will be run with SPMD parallelization enabled, and this requires setting up MPI resources and possibly dealing with batch queues. These issues will be addressed briefly in the Section called *Sample Run Scripts*. But for a simple test in serial mode executed from an interactive shell, we only need to issue the following command:

```
% /work/user/cam_test/bld/cam >&! cam.log
```

The commandline above redirects STDOUT and STDERR to the file `cam.log`. The CAM logfile contains a substantial amount of information from all components that can be used to verify that the model is running as expected. Things like namelist variable settings, input datasets used, and output datasets created are all echoed to the log file. This is the first place to look for problems when a model run is unsuccessful. It is also very useful to include relevant information from the logfile when submitting bug reports.

Sample Run Scripts

Examples

This section provides a few examples of using **configure** and **build-namelist** to set up a variety of model runs.

Running CAM via the CCSM scripts

See CCSM-4.0 User's Guide[5].

Notes

1. http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/book1.html
2. http://www.ccsm.ucar.edu/models/ccsm4.0/index.html
3. http://www.unidata.ucar.edu/downloads/netcdf/
4. http://en.wikipedia.org/wiki/SPMD
5. http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/book1.html

Appendix A. The configure utility

The **configure** utility provides a flexible way to specify a particular configuration of CAM. By default it will produce the configuration files required to build the standard production version of CAM (currently FV dynamics at 1.9x2.5 horizontal resolution with 26 levels).

configure produces the files `Filepath` and `Makefile`. In addition, a configuration cache file (`config_cache.xml` by default) is written which contains the values of all the configuration parameters set by **configure**. The files produced by running **configure** are written to the directory where CAM will be built, which by default is the directory from which **configure** is executed, but can be specified to be elsewhere (see the `-cam_bld` option).

configure will optionally perform tests to validate that the Fortran compiler is operational and Fortran 90 compliant, and that the linker can resolve references to required external libraries (NetCDF and possibly MPI). These tests will point out problems with the user environment in a way that is much easier to understand than looking at the output from a failed build of CAM. We strongly recommend that the first time CAM is built on any new machine, **configure** should be invoked to execute these tests (see the `-test` option).

Options to configure

All configuration options can be specified using command line arguments to **configure** and this is the recommended practice. Options specified via command line arguments take precedence over options specified any other way.

At the next level of precedence a few options can be specified by setting environment variables. And finally, at the lowest precedence, many options have hard-coded defaults. Most of these are located in the files `$CAM_ROOT/models/atm/cam/bld/config_files/defaults_*.xml`. A few that depend on the values of other options are set by logic contained in the **configure** script (a Perl script). The hard-coded defaults are designed to produce the standard production configurations of CAM.

The configure script allows the user to specify compile time options such as model resolution, dynamical core type, additional compiler flags, and many other aspects. The user can type **`configure --help`** for a complete list of available options.

The following options may all be specified with either one or two leading dashes, e.g., `-help` or `--help`. The few options that can be expressed as single letter switches may not be clumped, e.g., `-h -s -v` may *NOT* be expressed as `-hsv`. When multiple options are listed separated by a vertical bar either version may be used.

CAM configuration

These options will have an effect whether running CAM as part of CCSM or running in a CAM standalone mode:

`-[no]age_of_air_trcs`

Switch on [off] age of air tracers. Default: on for waccm_phys, otherwise off.

`-chem` **`<name>`**

Build CAM with specified chemistry package [`waccm_mozart` | `waccm_ghg` | `trop_mozart` | `trop_ghg` | `trop_bam` | `super_fast_llnl` | `none`] (default: `none`)

`-co2_cycle`

This option is meant to be used with the `-ccsm_seq` option. Its use in standalone CAM is not supported. (default: not set)

`-comp_intf` [`mct` | `esmf`]

Specify the component interfaces (default: `mct`).

`-cppdefs` **`<string>`**

A string of user specified CPP defines. Appended to Makefile defaults. E.g. `-cppdefs '-DVAR1 -DVAR2'`. Note that a string containing whitespace will need to be quoted.

`-dyn` [`eul` | `sld` | `fv` | `homme`]

Build CAM with specified dynamical core. (default: `fv`).

`-edit_chem_mech`

Envokes CAMCHEM_EDITOR to allow the user to edit the chemistry mechanism file.

`-hgrid` **`<name>`**

Specify horizontal grid. For spectral grids use `nlat`x`nlon` where `nlat` and `nlon` are the number of latitude and longitude grid points respectively in the Gaussian grid. For FV grids use `dlat`x`dlon` where `dlat` and `dlon` are the grid cell size in degrees for latitude and longitude respectively. For the HOMME cubed sphere grid use `ne`x`np`.

`-nadv` **`<n>`**

Set total number of advected species to **`<n>`**.

`-nadv_tt` **`<n>`**

Set number of advected test tracers to **`<n>`**.

`-nlev` **`<n>`**

Set number of levels to **`<n>`**.

`-pcols` **`<n>`**

Set maximum number of columns in a chunk to **`<n>`**.

`-pergro`

Switch enables building CAM for perturbation growth tests.

`-phys` [`cam4` | `ideal` | `adiabatic`]

Physics package option (default: `cam4`).

`-prog_species` **`<list>`**

Comma separated list of prognostic mozart species packages. Currently available: `DST,SSLT,SO4,GHG,OC,BC,CARBON16`

`-usr_mech_infile` **`<name>`**

Pathname of the user supplied chemistry mechanism file.

`-waccm_phys`

Switch enables the use of WACCM physics in any chemistry configuration. The user does not need to set this if one of the waccm chemistry options is chosen.

SCAM configuration

`-camiop`

Configure CAM to generate an IOP file that can be used to drive SCAM. This switch only works with the Eulerian dycore.

`-scam`

Compiles model in single column mode. Only works with Eulerian dycore.

CAM parallelization

`-[no]smp`

Switch on [off] SMP parallelism (OpenMP).

`-[no]spmd`

Switch on [off] SPMD parallelism (MPI).

CICE decomposition

When CAM is running standalone with CICE the CICE decomposition must be explicitly set using the following options:

> **Note:** *** Either set all of `-cice_bsizex`, `-cice_bsizey`, `-cice_maxblocks` and `cice_decomptype` or set `-ntasks` and/or `-nthreads` which are used to determine defaults for the cice decomposition. ***

`-cice_bsizex` **`<n>`**

CICE block size in longitude dimension. This size must evenly divide the number of longitude points in the global grid.

`-cice_bsizey` **`<n>`**

CICE block size in latitude dimension. This size must evenly divide the number of latitude points in the global grid.

`-cice_maxblocks` **`<n>`**

Maximum number of CICE blocks per processor.

`-cice_decomptype` **`<name>`**

CICE decomposition type [`cartesian` | `spacecurve`].

`-ntasks` **`<n>`**

Number of MPI tasks. Setting ntasks > 0 implies -spmd. Use -nospmd to turn off linking with an MPI library. To configure for pure MPI specify "-ntasks N -nosmp". ntasks is used to determine default grid decompositions. Currently only used by CICE.

`-nthreads` **`<n>`**

Number of OpenMP threads per process. Setting nthreads > 0 implies -smp. Use -nosmp to turn off compilation of OMP directives. For pure OpenMP set "-nthreads N -nospmd" nthreads is used to determine default grid decompositions. Currently only used by CICE.

General options

`-cache` **`<name>`**

Name of output cache file (default: `config_cache.xml`).

`-cachedir` **`<dir>`**

Name of directory where output cache file is written (default: CAM build directory).

`-ccsm_seq`

Switch to specify that CAM is being built from within the CCSM scripts.

`-help | -h`

Print usage to STDOUT.

`-silent | -s`

Turns on silent mode - only fatal messages printed to STDOUT.

`-[no]test`

Switch on [off] testing of Fortran compiler and linking to external libraries.

`-verbose | -v`

Turn on verbose echoing of settings made by configure.

`-version`

Echo the repository tag name used to check out this CAM distribution.

Surface components

Options for surface components used in standalone CAM mode:

`-ice` [`cice` | `sice` | `csim4` | `none`]

Specify the sea ice component. Default: `cice`.

`-lnd` [`clm` | `slnd` | `none`]

Specify the land component. Default: `clm`.

`-ocn` [`docn`|`socn`| `dom`|`none`]

Specify ocean component. Default: `dom`.

CAM standalone build

Options for building CAM via standalone scripts:

`-cam_bld` **`<dir>`**

Directory where CAM will be built. This is where configure will write the output files it generates (Makefile, Filepath, etc...). Default: ./

`-cam_exe` **`<name>`**

Name of the CAM executable. Default: `cam`.

`-cam_exedir` **`<dir>`**

Directory where CAM executable will be created. Default: CAM build directory.

`-cc` **`<name>`**

User specified C compiler. Overrides Makefile default.

`-cflags` **`<string>`**

A string of user specified C compiler options. Appended to Makefile defaults.

`-debug`

Switch to turn on building CAM with compiler options for debugging.

`-defaults` **`<name>`**

Specify a configuration file which will be used to supply defaults instead of one of the `config_files/defaults_*.xml` files. This file is used to specify model configuration parameters only. Parameters relating to the build which are system dependent will be ignored.

`-esmf_libdir` **`<dir>`**

Directory containing ESMF library and the `esmf.mk` file.

`-fc` **`<name>`**

User specified Fortran compiler. Overrides Makefile default.

`-fflags` **`<string>`**

A string of user specified Fortran compiler flags. Appended to Makefile defaults. See `-fopt` to override optimization flags.

`-fopt` **`<string>`**

A string of user specified Fortran compiler optimization flags. Overrides Makefile defaults.

`-gmake` **`<name>`**

Name of the GNU make program on your system. Supply the absolute pathname if the program is not in your path (or fix your path). This is only needed by **configure** for running tests via the `-test` option.

`-lapack_libdir` **`<dir>`**

Directory containing LAPACK library.

`-ldflags` **`<string>`**

A string of user specified load options. Appended to Makefile defaults.

`-linker` **`<name>`**

User specified linker. Overrides Makefile default of FC.

`-mpi_inc` **`<dir>`**

Directory containing MPI include files.

`-mpi_lib` **`<dir>`**

Directory containing MPI library.

`-nc_inc` **`<dir>`**

Directory containing NetCDF include files.

`-nc_lib` **`<dir>`**

Directory containing NetCDF library.

`-nc_mod` **`<dir>`**

Directory containing NetCDF module files.

`-pnc_inc` **`<dir>`**

Directory containing PnetCDF include files.

`-pnc_lib` **`<dir>`**

Directory containing PnetCDF library.

`-target_os` **`<name>`**

Override the OS setting for cross platform compilation from the following list [`aix|irix|linux| bgl|bgp`]. Default: OS on which configure is executed as defined by the Perl $OSNAME variable.

`-usr_src` **`<dir1>[,<dir2>[,<dir3>[...]]]`**

Directories containing user source code.

Environment variables recognized by configure

The following environment variables are recognized by **configure**. Note that the command line arguments for specifying this information always takes precedence over the environment variables.

ESMF_LIBDIR

Directory containing the ESMF library.

INC_MPI

Directory containing the MPI include files.

INC_NETCDF

Directory containing the NetCDF include files.

INC_PNETCDF

Directory containing the PnetCDF include files.

LAPACK_LIBDIR

Directory containing the LAPACK library.

LIB_MPI
: Directory containing the MPI library.

LIB_NETCDF
: Directory containing the NetCDF library.

LIB_PNETCDF
: Directory containing the PnetCDF library.

MOD_NETCDF
: Directory containing the NetCDF module files.

Appendix B. The build-namelist utility

The **build-namelist** utility builds namelists which specify run-time details for CAM. These are written to a single file (by default, the file `atm_in` in the directory from which **build-namelist** is invoked).

The only required input for **build-namelist** is a configuration cache file produced by a previous invocation of **configure** (`config_cache.xml` by default). **build-namelist** looks at this file to determine the features of the CAM executable, such as the dynamical core and horizontal resolution, that affect the default specifications for namelist variables. The default values themselves are specified in the file `$CAM_ROOT/models/atm/cam/bld/namelist_files/namelist_defaults_cam.xml`, and in the use case files located in the directory `$CAM_ROOT/models/atm/cam/bld/namelist_files/use_cases/`.

The methods for setting the values of namelist variables, listed from highest to lowest precedence, are:

1. using specific command-line options, i.e., `-case` and `-runtype`,
2. using the `-namelist` option,
3. setting values in a file specified by `-infile`,
4. specifying a `-use_case` option,
5. setting values in the namelist defaults file.

The first four of these methods for specifying namelist variables are the ones available to the user without requiring code modification. Any namelist variable recognized by CAM can be modified using method 2 or 3. The final two methods represent defaults that are hard coded as part of the code base.

Options to build-namelist

To get a list of all available options, type **`build-namelist --help`**. Available options are also listed just below.

The following options may all be specified with either one or two leading dashes, e.g., `-help` or `--help`. The few options that can be expressed as single letter switches may not be clumped, e.g., `-h -s -v` may *NOT* be expressed as `-hsv`. When multiple options are listed separated by a vertical bar either version may be used.

`-case` **`<name>`**

Case identifier up to 80 characters. This value is used to set the case_name variable in the driver namelist. Default: `camrun`

`-cice_nl` **`<namelist>`**

Specify namelist settings for CICE directly on the commandline by supplying a string containing FORTRAN namelist syntax, e.g., **`-cice_nl "&ice histfreq=1 /"`**. This namelist will be passed to the invocation of the CICE build-namelist via its `-namelist` argument.

`-config` **`<filepath>`**

Read the specified configuration cache file to determine the configuration of the CAM executable. Default: `config_cache.xml`.

`-config_cice` **`<filepath>`**

Filepath of the CICE config_cache file. This filepath is passed to the invocation of the CICE **build-namelist**. Only specify this to override the default filepath which was set when the CICE **configure** was invoked by the CAM **configure**.

`-csmdata` **`<dir>`**

Root directory of CCSM input data. Can also be set by using the CSMDATA environment variable.

`-dir` **`<dir>`**

Directory where output namelist files for each component will be written, i.e., `atm_in`, `drv_in`, `ice_in`, `lnd_in` and `ocn_in`. Default: current working directory.

`-help | -h`

Print usage to STDOUT.

`-ignore_ic_date`

Ignore the date attribute of the initial condition files when determining the default.

`-ignore_ic_year`

Ignore just the year part of the date attribute of the initial condition files when determining the default.

`-infile` **`<filepath>`**

Specify a file containing namelists to read values from.

`-inputdata` **`<filepath>`**

Writes out a list of pathnames for required input datasets to the specified file.

`-namelist` **`<namelist>`**

Specify namelist settings directly on the commandline by supplying a string containing FORTRAN namelist syntax, e.g., **`-namelist "&atm stop_option='ndays' stop_n=10 /"`**

`-runtype` [`startup`|`continue`|`branch`]

Type of simulation. Default: `startup`.

`-silent | -s`

Turns on silent mode - only fatal messages issued.

`-test`

Enable checking that input datasets exist on local filesystem. This is also a convenient way to generate a list of the required input datasets for a model run.

`-use_case` **`<name>`**

Specify a use case.

`-verbose | -v`

Turn on verbose echoing of informational messages.

`-version`

Echo the source code repository tag name used to check out this CAM distribution.

Environment variables used by build-namelist

The environment variables recognized by **build-namelist** are presented below.

CSMDATA

Root directory of CCSM input data. Note that the commandline argument `-csmdata` takes precedence over the environment variable.

OMP_NUM_THREADS

If values of the specific variables that set the thread count for each component, i.e., **`atm_nthreads`**, **`cpl_nthreads`**, **`ice_nthreads`**, **`lnd_nthreads`**, or **`ocn_nthreads`**, are set via the `-namelist`, or `-infile` options, then these values have highest precedence. The OMP_NUM_THREADS environment variable has next highest precedence for setting any of the component specific thread count variables. Lowest precedence for setting these variables is the value of `nthreads` from the configure cache file.

Appendix C. CAM Namelist Variables

A CAM model run is controlled using the **build-namelist** facility described in Appendix B. The focus of this appendix is to provide a reference for the variables that may be set through the use of **build-namelist**. A searchable (or browsable) page is also available by following the "Search Namelist Variables" link under the Documentation section of the CAM home page[1].

Note: The table version of the variables is not yet ready.

Notes

1. http://www.ccsm.ucar.edu/models/ccsm4.0/cam