User’s Guide to the Community Atmosphere Model
CAM-5.1.1

Brian Eaton
NCAR
Table of Contents

Acknowledgments ............................................................................................................... v
1. Introduction .......................................................................................................................1
   Changes from previous release ..................................................................................1
   Getting Help -- Other User Resources .......................................................................2
   The CAM Web Page ........................................................................................................2
   The CAM Bulletin Board .............................................................................................2
   Reporting bugs .............................................................................................................2
2. Building and Running CAM ..........................................................................................5
   Sample Interactive Session ..........................................................................................6
   Configuring CAM for serial execution ........................................................................6
   Specifying the Fortran compiler ....................................................................................7
   Configuring CAM for parallel execution .......................................................................9
   Building CAM .............................................................................................................11
   Building the Namelist .................................................................................................11
   Acquiring Input Datasets ............................................................................................14
   Running CAM .............................................................................................................14
   Sample Run Scripts .....................................................................................................15
   Examples ......................................................................................................................15
   Running CAM via the CESM scripts ............................................................................15
3. Model Output ..................................................................................................................17
   Model History Files ....................................................................................................17
   Fields Output to History Files ....................................................................................17
A. The configure utility ......................................................................................................19
   How configure is called from the CESM scripts .......................................................19
   Options to configure ....................................................................................................19
   CAM configuration .......................................................................................................20
   SCAM configuration ....................................................................................................21
   CAM parallelization ....................................................................................................22
   CICE decomposition ....................................................................................................22
   General options ..........................................................................................................23
   Surface components ....................................................................................................23
   CAM standalone build ...............................................................................................24
   Environment variables recognized by configure .....................................................25
B. The build-namelist utility .............................................................................................27
   Options to build-namelist ............................................................................................28
   Environment variables used by build-namelist ..........................................................29
C. CAM Namelist Variables .............................................................................................31
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-5.1.1.

The new model would not exist without the significant input from members of the CESM Atmospheric Model Working Group (AMWG) too numerous to mention. Phil Rasch (PNNL), Rich Neale (NCAR), Minghua Zhang (SUNY) and Leo Donner (GFDL), were co-chairs of the AMWG during part or all of the development of CAM-5.1.1.

We would like to acknowledge the substantial contributions to the CAM-5.1.1 effort from the National Science Foundation, the Department of Energy, the National Oceanic and Atmospheric Administration, and the National Aeronautics and Space Administration.
Acknowledgments
Chapter 1. Introduction

The Community Atmosphere Model version CAM-5.1.1 is released as the atmosphere component of the Community Earth System Model version CESM-1.0.4. It is the latest in a series of global atmosphere models whose development is guided by the Atmosphere Model Working Group\(^1\) (AMWG) of the Community Earth System Model\(^2\) (CESM) project. CAM is used as both a standalone model and as the atmospheric component of the CESM. 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 a standalone configuration that is being used. When CAM is coupled to active ocean and sea ice models then we refer to the model as CESM.

CAM provides a framework for running the "Whole Atmosphere" configurations; WACCM, and WACCM-X. To run CAM in a WACCM or WACCM-X configuration the user is referred to the CESM-1.0 User's Guide\(^3\).

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 the CCSM. One of the most significant software changes in CAM-4.0 was 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 (this also depended on the complete rewriting of the CCSM coupler to support sequential execution of the components). Hence, for the CESM1 model, just as for CCSM4 before it, it is accurate to say that a CAM standalone configuration is nothing more than a special configuration of CESM 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 CESM it can be run using the CESM scripts. This is done by using one of the "F" compsets and is described in the CESM-1.0 User's Guide\(^4\). The main advantage of running CAM via the CESM scripts is to leverage the high level of support that those scripts provide for doing production runs of predefined experiments on supported platforms. The CESM 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 a lot 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 `config` utility and any runtime configuration can be set up with appropriate arguments to CAM’s `build-namelist` utility. Issues that are specific to running CAM from the CESM scripts will not be discussed in this guide. Rather we focus on issues that are independent of which scripts are used to run CAM, although there is some attention given in this guide to the construction of simple scripts designed for running CAM in its standalone mode.

Changes from previous release

This information is available from the CESM-1.0 home page\(^5\). The changes for CAM-5.1.1 are categorized as changes to the CAM component of CESM-1.0.4.

- New features in CAM-5.1.1\(^6\) which includes notes on code cleanup and refactoring as well as bug fixes. The most important new features are the addition of the WACCM-X configuration and a much more efficient default configuration for running the COSP simulator.
Chapter 1. Introduction

- Summary of answer changes. The only significant answer changes are due to a bug fix in the molecular diffusion code used by the WACCM configurations. This fix changes the climate of WACCM simulations.
- There is a listing of known problems on the Release Notes page. In addition the following are CAM specific known bugs:
  - “-chem trop_mam7” currently not working. This is a side effect of a partial code refactoring in modal_aer_opt.

Getting Help -- Other User Resources

The CAM Web Page

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

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. 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/CESM if CAM was obtained as part of a CCSM or CESM distribution).
2. The architecture on which the code was built. Include relevant information such as the Fortran compiler, MPI library, etc.
3. The `configure` command line. If it is this command that is failing, then report the output from this command. It can also be very useful to run this command with the `-v` option to turn on verbose output.
4. The `build-namelist` command line. If it is this command that is failing, then report the output from this command. It can also be very useful to run this command with the `-v` option to turn on verbose output.
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 CAM is a research tool, and not all features contained in the code base are supported.
Notes

1. /working_groups/Atmosphere/
2. /models/cesm1.0/
3. /models/cesm1.0/cesm/cesm_doc_1_0_4/book1.html
4. /models/cesm1.0/cesm/cesm_doc_1_0_4/book1.html
5. /models/cesm1.0/
6. /models/cesm1.0/tags/cesm1_0_4/whatsnew.html
7. /models/cesm1.0/tags/cesm1_0_4/answerchanges.html
8. /models/cesm1.0/tags/index.html
9. /models/cesm1.0/cam
Chapter 1. Introduction
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 CESM scripts (see CESM-1.0 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. CAM-5.1.1 is distributed with CESM-1.0.4. To obtain the source code go to the section "Acquiring the Code" on the CESM Home Page\(^2\). When we refer to the root of the CAM source tree, this is the same directory as the root of the CESM 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 4.1.3 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\(^3\). We have updated the required NetCDF library version from 3.6 to 4.1.3 due to a recently discovered bug which affects all previous versions of the NetCDF library. The bug only occurs in special circumstances that are not that easy to replicate, however the result is the corrupt files are silently created. A more complete description of the bug is here\(^4\).
- 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 Spectral Element), 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 (or "launches") MPI executables. When running with 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 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).

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, using a UNIX shell syntax, 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 $CCSMROOT since the root directory of the CESM distribution is the same as the root of the CAM distribution which is contained within it.

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 command-line 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 (see the example just below). The argument `-dyn fv` specifies using the FV dynamical core which is the default for CAM5, but we recommend always adding the dynamical core (dycore for short) 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 shared 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. Also the method of launching an MPI job is system dependent. But the build and static linking will still be tested.

```sh
% $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 -ntasks 1 -nthreads 1 -cache config_cache_cice.xml \
  -cachedir /work/user/cam_test/bld
configure done.
creating /work/user/cam_test/bld/Filepath
creating /work/user/cam_test/bld/Makefile
creating /work/user/cam_test/bld/config_cache.xml
Looking for a valid GNU make... using gmake
Testing for Fortran 90 compatible compiler... using pgf90
Test linking to NetCDF library... ok
CAM configure done.
```

The next three lines of output inform the user of the files being created by `configure`. All these files 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.

The output from the `-test` option tells us that `gmake` is a GNU Make on this machine; that the Fortran compiler is `pgf90`; and that code compiled with the Fortran compiler can be successfully linked to the NetCDF library. The CAM Makefile is the place where the default compilers are specified. On Linux systems the default is `pgf90`. Finally, since this is a serial configuration no test for linking to the MPI library was done.

**Specifying the Fortran compiler**

In the previous section the `configure` command was issued without specifying which Fortran compiler to use. For that to work we were depending on the CAM makefile to select a default compiler. One of the differences between the CAM standalone build and a build using the CESM scripts is that CAM provides a generic makefile which provides defaults based on the operating system name (as determined by the Perl internal variable `$OSNAME`), while the CESM scripts require the user to specify the machine (and compiler if the machine supports more than one) as an argument to the `create_newcase` command.
Chapter 2. Building and Running CAM

The CAM makefile currently recognizes the following operating systems and compilers.

AIX
   xlf90_r, mpxlf90_r

Linux
   pgf90 (this is the default)
   lf95
   ifort
   pathf90 (has had minimal testing)
   g95 (not tested)

Darwin
   xlf90_r

BGL
   mpxlf90

BGP
   mpxlf90_r

The above list contains two IBM Blue Gene machines; BGL and BGP. The executables on these machines are produced by cross compilation and hence the configure script is not able to determine the machine for which the build is intended. In this case the user must supply this information to configure by using the -target_os option with the values of either bgl or bgp.

On a Linux platform several compilers are recognized with the default being pgf90. It is assumed that the compiler to be used is in the user’s path (i.e., in one of the directories in the PATH environment variable). If it isn’t then the -test option will issue an error indicating that the compiler was not found.

Suppose for example that one would like to use the Intel compiler on a local Linux system. The CAM makefile recognizes ifort as the name of the Intel compiler. To invoke this compiler use the -fc argument to configure. The following example illustrates the output you get when the compiler you ask for isn’t in your PATH (assuming the -test option is used):

% $camcfg/configure -fc ifort -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 -ntasks 1 -nthreads 1 -cache config_cache_cice.xml \ 
   -cachedir /work/user/cam_test/bld
configure done.
creating /work/user/cam_test/bld/Filepath
creating /work/user/cam_test/bld/Makefile
creating /work/user/cam_test/bld/config_cache.xml
Looking for a valid GNU make... using gmake
Testing for Fortran 90 compatible compiler... **** FAILED ****
Issued the command:
gmake -f /work/user/cam_test/bld/Makefile test_fc 2>&1

The output was:

****
gmake: ifort: Command not found
gmake: *** [test_fc.o] Error 127

8
Some verbose output was eliminated for clarity. But the final lines of output let the user know that the ifort compiler was not found. This means that the PATH environment variable has not been correctly set. The first thing to try is to verify the directory that contains the compiler, and then to prepend this directory name to the PATH environment variable.

Another instance where the user needs to supply information about the Fortran compiler type to configure is when the compiler is being invoked by a wrapper script. A common example of this is using the mpi90 command to invoke the Fortran compiler that was used to build the MPI libraries. This facilitates correct compilation and linking with the MPI libraries without the user needing to add the required include and library directories, or library names. The same benefit is provided by the fttn wrapper used on Cray XT and XE systems. In the usual case that a Linux OS is being used, since the CAM makefile will not recognize these compiler names, it will assume that the default compiler is being used, and thus will supply compiler arguments that are appropriate for pgf90. The compilation will fail if pgf90 is not the compiler being invoked by the wrapper script (invoking configure with the -test option is a good way to catch this problem). The way to specify which Fortran compiler is being invoked by a wrapper script is via the -fc_type argument to configure. This argument takes one of the values pgi, lahey, intel, or pathscale.

The -fc_type argument to configure could also be used if it is desired to use a Fortran compiler that isn’t in the above list of recognized compilers, but the compiler is one of the types listed above. For example, to use the pgf95 compiler the configure commandline would contain "-fc pgf95 -fc_type pgi".

Note: We have not yet ported CAM to work with the gfortran compiler.

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), 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. Prior to the introduction of CICE as the sea ice model SPMD was turned on using the -spmd option. But if we try that now 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=
```

A requirement of the CICE model is that its 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. In order for CICE’s configure to set the decomposition it needs to know how much parallelism is going to be used. This information is provided by specifying the number of MPI tasks that the job will use via setting the -ntasks argument.
Chapter 2. Building and Running CAM

Note: The default CICE decomposition can be overridden by setting it explicitly using the `configure` options provided for that purpose.

When running CAM in SPMD mode the build procedure must be able to find the MPI include files and library. The recommended method for doing this is to use scripts provided by the MPI installation to invoke the compiler and linker. On Linux systems a common name for this script is `mpif90`. The CAM Makefile does not currently use this script by default on Linux platforms, so the user must explicitly specify it on the `configure` command line using the `-fc` argument:

```
% $camcfg/configure -fc mpif90 -dyn fv -hgrid 10x15 -ntasks 6 -nosmp -test
```

Issuing command to the CICE configure utility:

```
$CAM_ROOT/models/ice/cice/bld/configure -hgrid 10x15 -cice_mode prescribed \
   -ntr_aero 0 -ntasks 6 -nthreads 1 -cache config_cache_cice.xml \
   -cachedir /work/user/cam_test/bld
```

configure done.

```
creating /work/user/cam_test/bld/Filepath
creating /work/user/cam_test/bld/Makefile
creating /work/user/cam_test/bld/config_cache.xml
```

Looking for a valid GNU make... using gmake

Testing for Fortran 90 compatible compiler... using mpif90

Test linking to NetCDF library... ok

Test linking to MPI library... ok

CAM 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 `-nosmp` 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. Similarly to SPMD mode, prior to the introduction of CICE the SMP mode was turned on using the `-smp` option. But with CAM5 that will result in the same error from CICE that we obtained above from attempting to use `-smp`. 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 -nosmpmd -nthreads 6
```

Issuing command to the CICE configure utility:

```
$CAM_ROOT/models/ice/cice/bld/configure -hgrid 10x15 -cice_mode prescribed \
   -ntr_aero 0 -ntasks 1 -nthreads 6 -cache config_cache_cice.xml \
   -cachedir /work/user/cam_test/bld
```

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 10`
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. It is, of course, possible to execute build-namelist in the build directory since that’s where the cache file is and so you don’t need to specify to build-namelist where to find that file (it looks in the current working directory by default). But then, assuming you plan to run CAM in a different directory, all the files produced by build-namelist need to be copied to the run directory. If you’re running configure and build-namelist from a script, then you need to know how to generate the filenames for the files that need to be copied. For this reason it’s more robust to change to the run directory and execute build-namelist there. That way if there’s a change to the files that are produced, your script doesn’t break due to the files not all getting copied to the run 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 DOCN namelist to ./docn_ocn_in
Writing DOCN stream file to ./docn.stream.txt
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 ./docn_in
Writing CAM namelist to ./atm_in

Checking whether input datasets exist locally...
OK -- found depvel_file = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart/devel/devel_monthly.nc
OK -- found tracer_cnst_filelist = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/tracer_cnstfiles/oxid/oxid_1.9x2.5_L26_clim_list.c090805.txt
OK -- found tracer_cnst_datapath = /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/tracer_cnstfiles/oxid
OK -- found ext_frc_specifier for SO2 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found ext_frc_specifier for bc_a1 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found ext_frc_specifier for num_a1 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found ext_frc_specifier for num_a2 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found ext_frc_specifier for soa_a1 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found ext_frc_specifier for soa_a2 -> /fs/cgd/csm/inputdata/atm/cam/physprops/modal_optics_3mode_c100507.nc
OK -- found rad_climate for P_so4_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/sulfate_rrtmg_c080918.nc
OK -- found srf_emis_specifier for SO2 -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for SO2 -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/aerocom_mam3_dms_surf_2000_c090129.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5_surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2_surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5 Surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2 Surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5 Surf_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_so2 Surf_2000_c090726.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_soag_1.5 Sur_2000_c100217.nc
OK -- found srf_emis_specifier for DMS -> /fs/cgd/csm/inputdata/atm/cam/chem/trop_mozart_aero/emis/ar5_mam3_s
Chapter 2. Building and Running CAM

OK -- found rad_climate for P_pom_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/ocpho_rrtmg_c101112.nc
OK -- found rad_climate for P_soa_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/ocphi_rrtmg_c100508.nc
OK -- found rad_climate for P_bc_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/bcpho_rrtmg_c100508.nc
OK -- found rad_climate for P_dst_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/dust4_rrtmg_c090521.nc
OK -- found rad_climate for P_ncl_a1:/fs/cgd/csm/inputdata/atm/cam/physprops/ssam_rrtmg_c100508.nc
OK -- found rad_climate for P_so4_a2:/fs/cgd/csm/inputdata/atm/cam/physprops/sulfate_rrtmg_c080918.nc
OK -- found rad_climate for P_soa_a2:/fs/cgd/csm/inputdata/atm/cam/physprops/ocphi_rrtmg_c100508.nc
OK -- found rad_climate for P_ncl_a2:/fs/cgd/csm/inputdata/atm/cam/physprops/ssam_rrtmg_c100508.nc
OK -- found rad_climate for P_dst_a3:/fs/cgd/csm/inputdata/atm/cam/physprops/dust4_rrtmg_c090521.nc
OK -- found rad_climate for P_ncl_a3:/fs/cgd/csm/inputdata/atm/cam/physprops/ssam_rrtmg_c100508.nc
OK -- found rad_climate for P_so4_a3:/fs/cgd/csm/inputdata/atm/cam/physprops/sulfate_rrtmg_c080918.nc

The first eight lines of output from build-namelist inform the user of the files that have been created. There are namelist files for ice component (ice_in), the land component (lnd_in), the data ocean component (docn_in, docn_ocn_in), the atmosphere component (atm_in), the driver (drv_in), and a file that is read by both the atmosphere and land components (drv_flds_in). There is also a "stream file" (docn.stream.txt) which is read by the data ocean component. Note that these file-names are hardcoded in the components and my not be changed without 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 CESM project then you will be able to download the file from the project’s 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 your specific filepath.

Expanding a bit on rerunning build-namelist: let’s say for example that the -test option informed you that the nndata file cami_0000-01-01_10x15_L30_c081013.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 nndata='/work/user/cam_test/cami_0000-01-01_10x15_L30_c081013.nc' /"
```

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

**Note:** This particular configuration of CAM which is using the default cam5 physics package requires that 57 datasets be specified in order to run correctly. Trying to manage namelists of that complexity by hand editing files is extremely error prone and is strongly discouraged. User modifications to the default namelist settings can be made in a number of ways while still letting build-namelist actually generate the final namelist. In particular, the -namelist, -infile, and -use_case arguments to build-namelist are all mechanisms by which the user can override default values or specify additional namelist variables and still allow build-namelist to do the error and consistency checking which makes the namelist creation process more robust.
Chapter 2. Building and Running CAM

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 CESM 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
- Issuer: SomeOrganizationalUnit, SomeOrganization, SomeCity, SomeState, --

(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 CESM scripts

See CESM-1.0 User’s Guide.

Notes

1. /models/cesm1.0/cesm/cesm_doc_1_0_4/book1.html
2. /models/cesm1.0/index.html
7. /models/cesm1.0/cesm/cesm_doc_1_0_4/book1.html
Chapter 2. Building and Running CAM
Chapter 3. Model Output

CAM produces a series of NetCDF format history files containing atmospheric grid-point data generated during the course of a run. It also produces a series of NetCDF format restart files necessary to continue a run once it has terminated successfully and a series of initial conditions files that may be used to initialize new simulations. The contents of these datasets are described below.

Model History Files

History files contain model data values written at specified frequencies during a run. Options are also available to record averaged, instantaneous, maximum, or minimum values on a field-by-field basis. If the user wishes to see a field written at more than one time frequency (e.g., daily, hourly), additional history files must be declared. This functionality is available via setting namelist variables.

History files may be visualized using various commercial or freely available tools. Examples include the NCAR Graphics package, FERRET, ncview, MATLAB, AVS, IDL, and Yorick. For a list of software tools for interacting with NetCDF files, view the link Software for Manipulating or Displaying NetCDF Data.

Fields Output to History Files

CAM is set up by default to output a set of fields to a single monthly average history file. There is a much larger set of available fields, known as the "master field list," from which the user can choose fields of interest to add to the history file via namelist settings. Both the set of default fields and the master field list depend on how CAM is configured. Due to the large number of fields we have chosen to make lists of fields for some standard configuration available via linked documents rather than to inline the lists here. Each of the field list documents is comprised of tables containing the lists of fields that are output by default as well as the master field list.

Note: The master field list tables may contain some fields that are not actually available for output. The presence of a field in the master field list is a necessary, but not sufficient condition that the corresponding field in the history file will contain valid data. This is because in some instances fields are added to the master field list (this is done in the source code) even though that field may not be computed in the configuration that is built (specified via the arguments to configure). When adding non-default fields to the history file it's important to check that the fields contain reasonable data before doing a long run.

Default History Fields and Master Field Lists

The following links provide tables of default and master field lists for some standard model configurations which are characterized by the values of the -dyn, -phys, and -chem arguments to configure.

- fv, cam4, none
- fv, cam4, trop_bam
- fv, cam5, trop_mam
- fv, cam4, waccm_mozart (use_case: waccm_2000_cam4)
- fv, cam4, super_fast_llnl (use_case: 2000_cam4_super_fast_llnl)
Chapter 3. Model Output

Notes

2. /models/cesm1.0/cam/docs/ug5_1/hist_flds_fv_cam4.html
3. /models/cesm1.0/cam/docs/ug5_1/hist_flds_fv_cam4_trop_bam.html
4. /models/cesm1.0/cam/docs/ug5_1/hist_flds_fv_cam5.html
5. /models/cesm1.0/cam/docs/ug5_1/hist_flds_fv_cam4_waccm.html
6. /models/cesm1.0/cam/docs/ug5_1/hist_flds_fv_cam4_super_fast_llnl.html
Appendix A. The configure utility

The configure utility provides a flexible way to specify any configuration of CAM. The best way to communicate to another user how you built CAM is to simply supply them with the configure commandline that was used (along with the source code version).

configure has two distinct operating modes which correspond to the two distinct ways of building CAM, i.e., either using the CESM scripts, or using CAM standalone scripts. By default configure runs in the mode used by the standalone scripts. In this mode configure is responsible for setting the filepaths and CPP macros needed to build not only CAM, but all the components of the standalone configuration including the land, sea ice, data ocean, and driver. In the mode used when building CAM from the CESM scripts configure is only responsible for setting the filepaths and CPP macros needed to build a library containing just the CAM component.

When configuring a build of standalone CAM, 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 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).

When configuring CAM for a build using the CESM scripts, configure doesn’t write a Makefile, but instead writes a file CCSM_cppdefs which is used by the CESM Makefile. Also, the Filepath file only contains paths for the CAM component.

In both modes configure is responsible for setting the correct filepaths and CPP macros to produce the desired configuration of CAM’s dynamical core, physics parameterizations and chemistry scheme. The options that are involved in making these choices are described in the Section called CAM configuration below. The subsequent sections describe options used by the CAM standalone scripts.

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).

How configure is called from the CESM scripts

The CESM scripts access CAM’s configure via the script $CAM_ROOT/models/atm/cam/bld/cam.cpl7.template. The cam.cpl7.template script acts as the interface between the CESM scripts and CAM’s configure and build-namelist utilities.

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.
Appendix A. The configure utility

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 CESM or running in a CAM standalone mode:*

- `-age_of_air_trcs`

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

- `-chem <name>`

  Build CAM with specified prognostic chemistry package [ waccm_mozart | waccm_mozart_v1 | waccm_ghg | trop_mozart | trop_ghg | trop_bam | trop_mam3 | trop_mam7 | super_fast_1lnl | super_fast_1lnl_mam3 | trop_strat_bam_v1 | none ] Default: trop_mam3 if the physics package is cam5, otherwise default is none.

- `-co2_cycle`

  This option is usually used with the `-ccsm_seq` option as part of the configuration for running biogeochemistry (BGC) compsets. It modifies the CAM configuration by increasing the number of advected constituents by 4. 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`

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

- `-hgrid <name>`

  Specify horizontal grid. For spectral grids use `nlatXnlon` where `nlat` and `nlon` are the number of latitude and longitude grid points respectively in the global Gaussian grid. For FV grids use `dlatXdlon` where `dlat` and `dlon` are the grid cell size in degrees for latitude and longitude respectively. For SE grids (cubed sphere) use `neXnp` where `ne` is the number of elements on an edge of the cube, and `np` is the number of Gauss points on the edge of an element.
Appendix A. The configure utility

-`microphys[mg | rk]`
  Microphysics package. Default: mg if the physics package is cam5, otherwise rk.

-`nadv <n>`
  Set total number of advected species to `<n>`. If `nadv` is set to a larger number than is required by the selected physics and chemistry schemes, then the remainder will automatically be used for test tracers. Default: set to the number required by the selected physics and chemistry schemes.

-`nadv_tt <n>`
  Set number of advected test tracers to `<n>`. Default: 0.

-`nlev <n>`
  Set number of vertical layers to `<n>`. Default: 30 if the physics package is cam5, ideal, or adiabatic. 26 if the physics package is cam4. 66 if the chemistry package is waccm_* and 81 if the `-waccmx` is used.

-`offline_dyn`
  Switch enables the use of offline driver for FV dycore. Default: not set.

-`pbl [uw | hb | hbr]`
  PBL package. Default: uw if the physics package is cam5, otherwise hb.

-`pcols <n>`
  Set maximum number of columns in a chunk to `<n>`. Default: 16.

-`pergro`
  Switch enables building CAM for perturbation growth tests. Only valid with cam3 and cam4 physics packages.

-`phys [cam3 | cam4 | cam5 | ideal | adiabatic]`

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

-`rad [rrtmg | camrt]`
  Radiation package. Default: rrtmg if the physics package is cam5, otherwise camrt.

-`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 wacmx chemistry options is chosen.

-`waccmx`
  Build CAM/WACCM with WACCM upper Thermosphere/Ionosphere extended package.
Appendix A. The configure utility

**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**

- **ntasks <n>**
  This option must be used to specify SPMD parallelism when the CICE component is present. <n> is the 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 by CICE to determine default grid decompositions which must be specified at build time.

- **nthreads <n>**
  This option must be used to specify SMP parallelism when the CICE component is present. <n> is the 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 -nosmp" nthreads is used by CICE to determine default grid decomposition which must be specified at build time.

- `[no]spmd`
  Switch on [off] SPMD parallelism (MPI). This option can be used when building a model that doesn’t contain CICE. It allows building an executable that is valid for any task count.

- `[no]smp`
  Switch on [off] SMP parallelism (OpenMP). This option can be used when building a model that doesn’t contain CICE. It allows building an executable that is valid for any thread count.

**CICE decomposition**

When CAM is running standalone with CICE the default CICE decomposition is determined from the values of the **ntasks** and **nthreads** arguments. The user also has the ability to explicitly set the CICE decomposition using the following arguments:

**Note:** ***All four of these arguments must be set.***

- **cice_bsizex <n>**
  CICE block size in longitude dimension. This size must evenly divide the number of longitude points in the global grid.
Appendix A. The configure utility

-\texttt{-cice\_bsizey} <\texttt{n}>
  
  CICE block size in latitude dimension. This size must evenly divide the number of latitude points in the global grid.

-\texttt{-cice\_maxblocks} <\texttt{n}>
  
  Maximum number of CICE blocks per processor.

-\texttt{-cice\_decomptype} <\texttt{name}>
  
  CICE decomposition type \{cartesian | spacecurve | roundrobin\}.

\textbf{General options}

-\texttt{-cache} <\texttt{name}>
  
  Name of output cache file. Default: \texttt{config\_cache.xml}.

-\texttt{-cachedir} <\texttt{dir}>
  
  Name of directory where output cache file is written. Default: CAM build directory.

-\texttt{-ccsm\_seq}
  
  Switch to specify that CAM is being built from within the CESM scripts. This produces Filepath and CCSM\_cppdefs files that contains only the paths and CPP macros needed to build a library for the CAM component.

-\texttt{-help} | \texttt{-h}
  
  Print usage to STDOUT.

-\texttt{-silent} | \texttt{-s}
  
  Turns on silent mode - only fatal messages printed to STDOUT.

-\texttt{[no]}\texttt{-test}
  
  Switch on [off] testing of Fortran compiler and linking to external libraries.

-\texttt{-verbose} | \texttt{-v}
  
  Turn on verbose echoing of settings made by configure.

-\texttt{-version}
  
  Echo the repository tag name used to check out this CAM distribution.

\textbf{Surface components}

\emph{Options for surface components used in standalone CAM mode:}

-\texttt{-clm\_vocsrc}
  
  Switch on VOC emissions in CLM. Default: off.

-\texttt{-ice} [\texttt{cice} | \texttt{sice} | \texttt{csim4} | \texttt{none}]
  
  Specify the sea ice component. Default: cice.

-\texttt{-lnd} [\texttt{clm} | \texttt{slnd} | \texttt{none}]
  
  Specify the land component. Default: clm.
Appendix A. The configure utility

-ocn [docn | socn | dom | none]
  Specify ocean component. Default: docn.

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. Default: set in Makefile depending on the OS and the
  Fortran compiler type.

-cflags <string>
  A string of user specified C compiler options appended to the default options
  set in Makefile.

-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. If this option is specified
  then the external ESMF library will be used in place of the ESMF-WRF time
  manager code which is provided in the CESM source distribution.

-fc <name>
  User specified Fortran compiler. Default: set in Makefile depending on the OS.

-fc_type [pgi | lahey | intel | pathscale]
  Type of the Fortran compiler. This argument is used in conjunction with the
  -fc argument when the name of the fortran compiler refers to a wrapper script (e.g.,
  mpif90 or ftn). In this case the user needs to specify the type of Fortran compiler
  that is being invoked by the wrapper script. Default: set in Makefile depending
  on the OS.

-fflags <string>
  A string of user specified Fortran compiler options appended to the default op-
  tions set in the Makefile. See -fopt to override optimization flags.
Appendix A. The **configure** utility

- **-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 \texttt{-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. Default: use the Fortran compiler.

- **-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.

- **-rad_driver**
  Build CAM with the offline radiation driver. This produces an executable that can only be used for offline radiation calculations.

- **-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 (and on occasion other types of input files) which specify run-time details for CAM and the components it’s running with in standalone mode. When executed from the CESM scripts it only produces a namelist file for the CAM component (in the file atm_in), and a namelist file for control of dry deposition which is shared by CAM and CLM (in the file drv_flds_in).

The task of constructing a correct namelist has become extremely complex due to the large number of configurations supported by CAM. Editing namelists by hand is an extremely fragile process due to the number of variables that need to be set, and to the many interdependencies among them. We do not recommend editing namelists by hand. All customizations of the CAM namelist are possible by making use of the command line options.

Some of the important features of build-namelist are:

- All valid namelist variables are known to build-namelist. So an invalid variable specified by the user (supplied either by the -infile or -namelist options) will cause build-namelist to fail with an error message telling which namelist variable is invalid. This is a big improvement over a runtime failure caused by an invalid variable which typically gives no hint as to which variable caused the problem.

- In addition to knowing all valid variable names and their types, build-namelist also knows which namelist group each variable belongs to. This means that the user only needs to specify variable names to build-namelist and not the group names. The -infile and -namelist options still require valid namelist syntax as input, but the group name is ignored. So all variables can be put in a single group with an arbitrary name, for example, "&xxx ... /".

- Since build-namelist knows all namelist variables specified by the user it is able to do consistency checking. In general however, build-namelist assumes that the user is the expert and will not override a user specification unless there is a major inconsistency, for example if variables have been set to use parameterizations which can not be run at the same time.

- All configurations have namelist variables that must be specified, and build-namelist has a mechanism to provide default values for these variables. When an appropriate default value cannot be found then build-namelist will fail with an informative message.

- When running a configuration for the first time there are often many input datasets that may not be in the local input data directory. In order to facilitate getting the required datasets build-namelist has an option, -test, that can be used to produce a complete list of required datasets and report status of whether or not they are present in the local directory. This list can then be used to obtain the needed datasets from the CESM SVN input data repository.

One 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 other required input for build-namelist is the root directory for the input datasets. This is required since nearly all input files are specified using absolute filepaths, but the locations of the files are stored relative to the root directory. It is expected that the actual location of the root directory is something that will be
Appendix B. The **build-namelist** utility

resolved at runtime. The way this is done is to either specify it using the `-csmdata` argument, or to set the environment variable CSMDATA.

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 `-runttype`,
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.
Appendix B. The build-namelist utility

- 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 CESM input data. Note that the commandline argument -csmdata takes precedence over the environment variable.
Appendix B. The **build-namelist** utility

**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 here¹, or by following the "Search Namelist Variables" link under the Documentation section of the CAM home page².

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

Notes

1. /cgi-bin/eaton/namelist/nldef2html-cam5_1_1
2. /models/cesm1.0/cam