

Community Climate System Model  
National Center for Atmospheric Research, Boulder, CO

# CCSM3.0 User's Guide

*Mariana Vertenstein, Tony Craig, Tom Henderson, Sylvia Murphy,  
George R Carr Jr and Nancy Norton*

[www.cesm.ucar.edu](http://www.cesm.ucar.edu)

June 25, 2004

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	CCSM overview . . . . .	4
1.2	CCSM model components . . . . .	4
1.2.1	Active atmospheric component . . . . .	5
1.2.2	Active land component . . . . .	5
1.2.3	Active ocean component . . . . .	5
1.2.4	Active sea-ice component . . . . .	5
1.3	Paleo-climate related issues . . . . .	6
1.3.1	Component Sets . . . . .	6
1.3.2	Component Resolutions . . . . .	7
1.4	Upgrades from previous releases . . . . .	9
1.5	Climate validations . . . . .	9
<b>2</b>	<b>Requirements</b>	<b>10</b>
2.1	Software Requirements . . . . .	10
2.2	Supported Machines . . . . .	10
2.3	Memory, disk space and simulation time limitations . . . . .	10
2.4	NCAR specific considerations . . . . .	11
2.4.1	Roy and the firewall . . . . .	11
2.4.2	Fileserver . . . . .	11
2.4.3	sftp,scp,ssh . . . . .	11
<b>3</b>	<b>Obtaining code, scripts and input data</b>	<b>12</b>
3.1	Downloading the model . . . . .	12
3.2	Untarring the source code and scripts . . . . .	12
3.3	Untarring the input data . . . . .	14
3.4	Testing the downloaded source code and data files . . . . .	15
<b>4</b>	<b>Creating cases (preparing model runs)</b>	<b>16</b>
4.1	Scripts usage overview . . . . .	16
4.2	<b>create_newcase</b> . . . . .	17
4.3	Resolved scripts . . . . .	18
4.4	<b>configure</b> . . . . .	18
4.4.1	<b>configure -mach</b> . . . . .	19
4.4.2	<b>configure -cleanmach and configure -cleanall</b> . . . . .	19
4.4.3	<b>configure -addmach</b> . . . . .	19
4.5	Environment variables . . . . .	20
4.6	Environment variables in <code>env_conf</code> . . . . .	21
4.7	Environment variables in <code>env_run</code> . . . . .	23
4.8	Environment variables in <code>env_mach.\$MACH</code> . . . . .	27
4.8.1	<code>env_mach.\$MACH</code> tasks/threads . . . . .	27
4.8.2	<code>env_mach.\$MACH</code> general machine specific settings . . . . .	27
4.8.3	<code>env_mach.\$MACH</code> input data prestaging . . . . .	28
4.8.4	<code>env_mach.\$MACH</code> output data short-term archiving environment variables . . . . .	28
4.8.5	<code>env_mach.\$MACH</code> output data long-term archiving . . . . .	28
4.9	Scripts design for developers . . . . .	29
4.10	General script tools . . . . .	30
4.11	Script generation tools . . . . .	31
4.11.1	Script validation tools . . . . .	31
4.11.2	Input/output data movement . . . . .	32

<b>5</b>	<b>Building, Running and Input/Output Data</b>	<b>33</b>
5.1	Model input	33
5.2	Building CCSM3	33
5.2.1	Building summary	33
5.2.2	<b>Gmake</b> details	34
5.2.3	User modified source code	35
5.3	Running CCSM3	35
5.4	Initialization types	36
5.4.1	Startup runs	37
5.4.2	Hybrid runs	38
5.4.3	Branch runs	38
5.4.4	Prestaging input datasets for hybrid or branch runs	39
5.5	Model output	39
5.5.1	Short-term archiving	40
5.5.2	Long-term archiving	41
<b>6</b>	<b>CCSM3 Use Cases</b>	<b>42</b>
6.1	Setting up a startup run	42
6.2	Setting up a branch run	42
6.3	Setting up a hybrid run	43
6.4	Setting up a production run	44
6.5	Monitoring a production run	45
6.6	Diagnosing a production run failure	45
6.7	Restarting a production run	46
6.8	Handing off a production run to another person	46
6.9	Generating IPCC configuration runs	46
6.10	Adding a new machine to <code>\$CCSMROOT/</code>	47
6.11	Modifying CCSM source code	48
6.12	Modifying input datasets	49
<b>7</b>	<b>Testing</b>	<b>50</b>
7.1	Features to be tested	50
7.2	Testing script	51
7.2.1	<b>create_test</b>	51
7.2.2	Using <b>create_test</b>	52
7.2.3	Available tests	53
7.3	Common <b>create_test</b> use cases	53
7.3.1	Validation of an Installation	54
7.3.2	Verification of Source Code Changes	55
<b>8</b>	<b>Performance</b>	<b>57</b>
8.1	Compiler Optimization	57
8.2	Model parallelization types	57
8.3	Load Balancing CCSM3	58
<b>9</b>	<b>Troubleshooting (Log Files, Aborts and Errors)</b>	<b>61</b>
9.1	Error messages and log files	61
9.2	General Issues	61
9.3	Model time coordination	62
9.4	POP ocean model non-convergence	62
9.5	CSIM model failures	62
9.6	CAM Courant limit warning messages	62

<i>LIST OF TABLES</i>	3
-----------------------	---

9.7	CAM model stops due to non-convergence	62
9.8	T31 instability onset	63
<b>10</b>	<b>Post Processing</b>	<b>64</b>
10.1	Software Overview	64
10.1.1	The netCDF Operators (NCO)	64
10.1.2	The NCAR Command Language (NCL)	64
10.1.3	Component Model Processing Suite (CMPS)	65
10.1.4	Diagnostic Packages	65
10.1.5	Commercial Tools	66
10.2	Post Processing Strategies	66
10.2.1	Selected Variable File Concatenation	66
10.2.2	Creating Climatologies	66
10.2.3	Atmospheric Hybrid to Pressure Level Interpolation	67
10.2.4	POP remapping	67
<b>11</b>	<b>Glossary</b>	<b>68</b>

## List of Tables

1	CCSM3 Model Components	4
1	CCSM3 Model Components	5
2	CCSM3.0 resolution-component set combinations	8
3	CCSM3 Supported Machines/Platforms	10
4	inputdata/ sub-directories	14
4	inputdata/ sub-directories	15
5	<b>env_conf, env_run and env_mach.\$MACH variables</b>	20
5	<b>env_conf, env_run and env_mach.\$MACH variables</b>	21
6	CCSM3 Components Parallelization	57
7	netCDF Operators	64
8	Component Model Processing Suite	65
9	Publicly available diagnostics	65

# 1 Introduction

## 1.1 CCSM overview

The Community Climate System Model (CCSM) is a coupled climate model for simulating the earth’s climate system. Composed of four separate models simultaneously simulating the earth’s atmosphere, ocean, land surface and sea-ice, and one central coupler component, the CCSM allows researchers to conduct fundamental research into the earth’s past, present and future climate states.

The CCSM project is a cooperative effort among US climate researchers. Primarily supported by the National Science Foundation (NSF) and centered at the National Center for Atmospheric Research (NCAR) in Boulder Colorado, the CCSM project enjoys close collaborations with the US Department of Energy and the National Air and Space Administration. Scientific development of the CCSM is guided by the CCSM working groups, which meet twice a year. The main CCSM workshop is held each year in June to showcase results from the various working groups and coordinate future CCSM developments among the working groups. More information on the CCSM project, such as the management structure, the scientific working groups, downloadable source code and online archives of data from previous CCSM experiments, can be found on the CCSM website

[www.cesm.ucar.edu](http://www.cesm.ucar.edu).

Help is available via email at

[cesm@ucar.edu](mailto:cesm@ucar.edu).

Additional CCSM3 documentation, including component user and science guides is available at the CCSM3 website at:

[www.cesm.ucar.edu/models/ccsm3.0](http://www.cesm.ucar.edu/models/ccsm3.0) .

## 1.2 CCSM model components

The CCSM consists of four dynamical geophysical models linked by a central coupler. Each model contains “active”, “data”, or “dead” component versions allowing for a variety of “plug and play” combinations. The active dynamical models consume substantial amounts of memory and CPU time and produce large volumes of output data. The data-cycling models (data models), on the other hand, are small, simple models which simply read existing datasets that were previously written by the dynamical models and pass the resulting data to the coupler. These data-cycling components are very inexpensive to run and produce no output data. For these reasons they are used for both test runs and certain types of model simulation runs. Currently, the data models run only in serial mode on a single processor. The dead models are simple codes that facilitate system testing. They generate unrealistic forcing data internally, require no input data and can be run on multiple processors to simulate the software behavior of the fully active system.

The CCSM components can be summarized as follows:

Table 1: CCSM3 Model Components

Model	Model Name	Component Name	Component Version	Type
atmosphere	atm	cam	cam3	active
atmosphere	atm	datm	datm6	data
atmosphere	atm	latm	latm6	data
atmosphere	atm	xatm	dead	dead
land	lnd	clm	clm3	active
land	lnd	dlnd	dlnd6	data
land	lnd	xlnd	dead	dead

Table 1: CCSM3 Model Components

ocean	ocn	pop	ccsm_pop_1_4	active
ocean	ocn	docn	docn6	data
ocean	ocn	xocn	dead	dead
sea-ice	ice	csim	csim5	active
sea-ice	ice	dice	dice6	data
sea-ice	ice	xice	dead	dead
coupler	cpl	cpl	cpl6	active

During the course of a CCSM run, the four non-coupler components simultaneously integrate forward in time, periodically stopping to exchange information with the coupler. The coupler meanwhile receives fields from the component models, computes, maps and merges this information and sends the fields back to the component models. By brokering this sequence of communication interchanges, the coupler manages the overall time progression of the coupled system. Each model has a full dynamical component, a data-cycling component (atm actually has 2 data cycling components), and a dead-version component. A CCSM component set is comprised of five model components - one component from each model. All model components are written primarily in FORTRAN 90.

### 1.2.1 Active atmospheric component

The dynamical atmosphere model is the Community Atmosphere Model (CAM), a global atmospheric general circulation model developed from the NCAR CCM3. The primary horizontal resolution is 128 longitudinal by 64 latitudinal points (T42) with 26 vertical levels. The hybrid vertical coordinate merges a terrain-following sigma coordinate at the bottom surface with a pressure-level coordinate at the top of the model.

### 1.2.2 Active land component

The Community Land Model, CLM, is the result of a collaborative project between scientists in the Terrestrial Sciences Section of the Climate and Global Dynamics Division (CGD) at NCAR and the CCSM Land Model Working Group. Other principal working groups that also contribute to the CLM are Biogeochemistry, Paleoclimate, and Climate Change and Assessment. The land model grid is identical to the atmosphere model grid.

### 1.2.3 Active ocean component

The ocean model is an extension of the Parallel Ocean Program (POP) Version 1.4.3 from Los Alamos National Laboratory (LANL). POP grids in CCSM are displaced-pole grids (centered at Greenland) at approximately 1-degree (gx1v3) and 3.6-degree (gx3v5) horizontal resolutions with 40 and 25 vertical levels, respectively. POP does not support a slab ocean model (i.e. SOM) as is supported by the stand-alone atmosphere model (CAM).

### 1.2.4 Active sea-ice component

The sea-ice component of CCSM is the Community Sea-Ice Model (CSIM). The sea-ice component includes the elastic-viscous-plastic (EVP) dynamics scheme, an ice thickness distribution, energy-conserving thermodynamics, a slab ocean mixed layer model, and the ability to run using prescribed ice concentrations. It is supported on high- and low-resolution Greenland Pole grids, identical to those used by the POP ocean model.

### 1.3 Paleo-climate related issues

CCSM3.0 requires modifications to run paleo-climate simulations. Tools were generated to facilitate these types of modifications in CCSM2.0. These tools have not yet been ported to CCSM3.0. In addition, CCSM3.0 has new physical parameterizations (specifically in CAM) that are relevant to only modern day climates. Users who desire to use CCSM3.0 for paleo-climate simulations are encouraged to first contact Zav Kothavala ([zav@ucar.edu](mailto:zav@ucar.edu)).

#### 1.3.1 Component Sets

The release version contains the following components:

- atmosphere:
  - cam (cam3) : fully active atmospheric model
  - datm (datm6): standard data atmospheric model
  - latm (latm6): climatological-data atm model
  - xatm (dead) : dead atmospheric model
- land:
  - clm (clm3) : fully active land model
  - dlnd (dlnd6): standard data land model
  - xlnd (dead) : dead land model
- ocean:
  - pop (pop) : fully active ocean model
  - docn (docn6): standard data ocean model
  - xocn (dead) : dead ocean model
- ice:
  - csim (csim5) : fully active ice model
  - dice (dice6): standard data ice model
  - xice (dead) : dead ice model
- coupler:
  - cpl (cpl6) : coupler

These components can be combined in numerous ways to carry out various science or software experiments. A particular mix of components is referred to as a **component set**. In general, there is a shorthand naming convention for component sets that are officially supported and which may be used “out of the box”. Users are not strictly limited to the following component set combinations. A user may define their own component set for a run by manually editing the `env_conf` file (see section 4.6).

```
A = datm,dlnd,docn,dice,cpl
B = cam, clm, pop,csim,cpl
C = datm,dlnd, pop,dice,cpl
D = datm,dlnd,docn,csim,cpl
G = latm,dlnd, pop,csim,cpl
H = cam,dlnd,docn,dice,cpl
I = datm, clm,docn,dice,cpl
K = cam, clm,docn,dice,cpl
L = latm,dlnd, pop,dice,cpl
M = latm,dlnd,docn,csim (mixed layer ocean mode),cpl
O = latm,dlnd,docn,dice,cpl
X = xatm,xlnd,xocn,xice,cpl
```

### 1.3.2 Component Resolutions

The following component resolutions are supported in CCSM3.0:

- atmosphere, land
  - T85 - gaussian grid, 256 longitudes, 128 latitudes [cam, clm]
  - T42 - gaussian grid, 128 longitudes, 64 latitudes [cam, clm, datm, dlnd]
  - T31 - gaussian grid, 96 longitudes, 48 latitudes [cam, clm, datm, dlnd]
  - 2x2.5 - type C grid, 144 longitudes, 91 latitudes [cam-FV dycore, clm]
  - T62 - gaussian grid, 192 longitudes, 94 latitudes [latm, dlnd]
- ocean, ice
  - gx1v3 - 320 longitudes, 384 latitudes
  - gx3v5 - 100 longitudes, 116 latitudes

The CCSM3 ocean and ice models must share a common horizontal grid. Presently, two different resolutions are officially supported in CCSM: gx1v3 and gx3v5. In both of these grids, the North Pole has been displaced into Greenland. gx1v3 is finer than gx3v5, and has a longitudinal resolution of approximately one degree. Its latitudinal resolution is variable, with finer resolution, approximately 0.3 degrees, near the equator. gx3v5 is the coarser grid, with a longitudinal resolution of 3.6 degrees. Its latitudinal resolution is variable, with finer resolution, approximately 0.9 degrees, near the equator.

In the ocean model, there are 40 vertical levels associated with the gx1v3 resolution, with level thickness monotonically increasing from approximately 10 to 250 meters. It is the combination of the horizontal grid, horizontal land mask, vertical grid, and bottom topography that collectively define the “gx1v3” resolution.

There are 25 vertical ocean-model levels associated with the gx3v5 resolution, with level thickness monotonically increasing from approximately 12 to 450 meters. The combination of the horizontal grid, horizontal land mask, vertical grid, and bottom topography is referred to collectively as the “gx3v5” resolution.

The CCSM3 atmosphere and land models must also share a common horizontal grid. Currently, four different resolutions are officially supported: T85, T42, T31 and 2x2.5. The first three correspond to gaussian grids, whereas the latter corresponds to a type C grid. All the grids are run in CAM with 26 vertical levels.

The above resolutions are supported “out of the box” in the following combinations:

- T85\_gx1v3
- T42\_gx1v3
- T42\_gx3v5
- T31\_gx3v5
- 2x2.5\_gx1v3
- T62\_gx1v3
- T62\_gx3v5

Finally, only certain combinations of components sets and model resolutions are supported by the scripts. The following table summarizes the allowed CCSM3.0 resolution-component set combinations.

Table 2: CCSM3.0 resolution-component set combinations

Comp-Set	Components	Resolution	Tested
A	datm,dlnd,docn,dice,cpl	T42_gx1v3	Yes
A	datm,dlnd,docn,dice,cpl	T42_gx3v5	No
A	datm,dlnd,docn,dice,cpl	T31_gx3v5	No
B	cam, clm, pop,csim,cpl	T85_gx1v3	Yes
B	cam, clm, pop,csim,cpl	T42_gx1v3	Yes
B	cam, clm, pop,csim,cpl	T42_gx3v5	No
B	cam, clm, pop,csim,cpl	T31_gx3v5	Yes
B	cam, clm, pop,csim,cpl	2x2.5_gx1v3	Yes
C	datm,dlnd, pop,dice,cpl	T42_gx1v3	No
C	datm,dlnd, pop,dice,cpl	T42_gx3v5	No
C	datm,dlnd, pop,dice,cpl	T31_gx3v5	No
D	datm,dlnd,docn,csim,cpl	T42_gx1v3	No
D	datm,dlnd,docn,csim,cpl	T42_gx3v5	No
D	datm,dlnd,docn,csim,cpl	T31_gx3v5	No
G	latm,dlnd, pop,csim,cpl	T62_gx1v3	Yes
G	latm,dlnd, pop,csim,cpl	T62_gx3v5	Yes
H	cam,dlnd,docn,dice,cpl	T85_gx1v3	No
H	cam,dlnd,docn,dice,cpl	T42_gx1v3	No
H	cam,dlnd,docn,dice,cpl	T42_gx3v5	No
H	cam,dlnd,docn,dice,cpl	T31_gx3v5	No
H	cam,dlnd,docn,dice,cpl	2x2.5_gx1v3	No
I	datm, clm,docn,dice,cpl	T42_gx1v3	No
I	datm, clm,docn,dice,cpl	T42_gx3v5	No
I	datm, clm,docn,dice,cpl	T31_gx3v5	No
K	cam, clm,docn,dice,cpl	T85_gx1v3	No
K	cam, clm,docn,dice,cpl	T42_gx1v3	No
K	cam, clm,docn,dice,cpl	T42_gx3v5	No
K	cam, clm,docn,dice,cpl	T31_gx3v5	No
K	cam, clm,docn,dice,cpl	2x2.5_gx1v3	No
K	cam, clm,docn,dice,cpl		
L	latm,dlnd, pop,dice,cpl	T62_gx1v3	Yes
L	latm,dlnd, pop,dice,cpl	T62_gx3v5	Yes
M	latm,dlnd,docn,csim(ML0*),cpl	T62_gx1v3	Yes
M	latm,dlnd,docn,csim(ML0*),cpl	T62_gx3v5	No
O	latm,dlnd,docn,dice,cpl	T62_gx1v3	No
O	latm,dlnd,docn,dice,cpl	T62_gx3v5	No

X	xatm,xlnd,xocn,xice,cpl	T85_gx1v3	No
X	xatm,xlnd,xocn,xice,cpl	T42_gx1v3	No
X	xatm,xlnd,xocn,xice,cpl	T42_gx3v5	No
X	xatm,xlnd,xocn,xice,cpl	T31_gx3v5	No
X	xatm,xlnd,xocn,xice,cpl	2x2.5_gx1v3	No
X	xatm,xlnd,xocn,xice,cpl	T62_gx1v3	Yes
X	xatm,xlnd,xocn,xice,cpl	T62_gx3v5	Yes

=====

\* MLO indicate Mixed Layer Ocean Mode

## 1.4 Upgrades from previous releases

The CCSM3.0 release contains significant upgrades from previous releases. The climate simulation capability has been improved and the model runs at new resolutions and on new grids. In addition, new platforms have been added and the CCSM scripts have been completely rewritten to be more easily used by both novice and expert users. They also make it much simpler for the user to add their own platforms to the script by separating out machine-specific settings. This provides the user with the ability to easily tailor the scripts for their particular machine environment.

High-, medium- and low-resolution versions of the CCSM components are supported in CCSM3.0. The higher resolution version (T85\_gx1v3) is best suited for simulating near-past, present-day and future climate scenarios, while the lower resolution option (T31\_gx3v5) is commonly used for paleoclimate research.

A much larger variety of platforms is now supported. Whereas CCSM2 was only supported on scalar architectures, CCSM3.0 is supported and runs effectively on both scalar and vector platforms. In addition, CCSM3.0 is now supported on several Linux clusters.

CCSM3.0 also includes the introduction of dead models which provide the ability to easily test the CCSM3 software infrastructure.

## 1.5 Climate validations

Although CCSM3.0 can be run “out of the box” for a variety of resolutions and component sets, it must be stressed that not all combinations of component sets, resolution and machines have been tested or have undergone full climate validations.

Long control runs have been carried out on the IBM systems at NCAR with the fully active CCSM components (component set B below) at three different resolutions: T85\_gx1v3, T42\_gx1v3, T31\_gx3v5. As a result, NCAR will only guarantee the scientific validity of runs using the above component set and resolutions on the IBM. No other combination of resolutions, component sets or machines are considered scientifically validated.

Model output from these long control runs will accompany the release. Users should be able to duplicate the climate of the released control runs using the CCSM3.0 source code on the NCAR IBM systems. Bit-for-bit duplication cannot be ensured due to post-release compiler changes that may occur. Users should carry out their own validations on any platform prior to doing scientific runs or scientific analysis and documentation.

## 2 Requirements

### 2.1 Software Requirements

The following tools and libraries are required in order to build and run the CCSM3.0 model:

- Compilers: Fortran 90, C
- Tools: gunzip, gmake, Perl5 or higher
- Libraries: MPI, netCDF

The above compilers, libraries and tools do not accompany this release. It is the user's responsibility to download them, install them, and if necessary modify their search path to include them.

### 2.2 Supported Machines

The following machines and corresponding platforms are supported at one of three different support levels in CCSM3.0 (denoted by RL).

- RL=1 corresponds to machines where climate validations have been performed.
- RL=2 corresponds to machines that have passed the CCSM restart test for the fully active component set (see 7).
- RL=3 corresponds to machines where the CCSM has been built successfully but where the resulting model may or may not have been run successfully, and where if the model ran, restart validation may or may not have been successful.
- RL=4 implies that this machine may be supported sometime in the future.

Table 3: CCSM3 Supported Machines/Platforms

Machine	Description	OS	Compiler	Network	Queue Software	RL
bluesky	IBM Pow4 8-way	AIX	IBM XL	IBM	Load Leveler	1
bluesky32	IBM Pow4 32-way	AIX	IBM XL	IBM	Load Leveler	1
blackforest	IBM Pow3 4-way	AIX	IBM XL	IBM	Load Leveler	1
cheetah	IBM Pow4 8-way	AIX	IBM XL	IBM	Load Leveler	1
cheetah32	IBM Pow4 32-way	AIX	IBM XL	IBM	Load Leveler	1
seaborg	IBM Pow3 16-way	AIX	IBM XL	IBM	Load Leveler	1
chinook	SGI	IRIX64	MIPS	NumaLink	NQS	2
jazz	Intel Xeon	Linux	PGI	Myrinet	PBS	2
anchorage	Intel Xeon	Linux	PGI	Gbit Ethernet	SPBS	3
bangkok	Intel Xeon	Linux	PGI	Gbit Ethernet	SPBS	3
lemieux	Alpha	OSF1	Compaq	Myrinet	PBS	3
moon	NEC Earth Sim	NEC	NEC	NEC	PBS	3
phoenix	Cray X1	Unicos/mp	Cray	CrayLink	PBS	4
calgory	Intel Xeon	Linux	PGI	InfiniBand	SPBS	4
rex	AMD Opteron	Linux	PGI	Myrinet	PBS	4
lightning	AMD Opteron	Linux	PGI	Myrinet	LSF	4

### 2.3 Memory, disk space and simulation time limitations

Users have numerous options for customizing the CCSM3.0 to suit their own research interests: various combinations of active, data, and dead components, seven possible resolutions, and multiple

combinations of platforms, compilers, and operating systems. This flexibility provides obvious benefits to researchers, but also presents a challenge when trying to quantify requirements such as memory, disk space, and simulation times.

For this reason, detailed documentation of such requirements is beyond the scope of this document. In the future, we plan to provide detailed requirements information from the CCSM web page. Until then, the following summary of the requirements from a typical T42\_gx1v3 CCSM3.0 model run at NCAR on the IBM machine "bluesky" may provide the user with a general idea of model requirements:

- Run Length: one model year
- Resolution: T42\_gx1v3
- History-File Volume: 6.5 Gbytes/model year
- Restart-File Volume: 0.9 Gbytes/model year
- Total CPUs: 104
- Simulation Years/Day: 7.5
- GAUs/Simulation Year: 85, regular queue

## 2.4 NCAR specific considerations

Supercomputing at NCAR has undergone significant changes since the last release of the CCSM. This section will summarize some of those changes. For the most up-to-date information, visit the following web site (gatekeeper password required):

[www.scd.ucar.edu/docs/access/internal/security.html](http://www.scd.ucar.edu/docs/access/internal/security.html)

### 2.4.1 Roy and the firewall

NCAR's supercomputers are now located behind a firewall. To login, it is necessary to first logon to roy.scd.ucar.edu using the gatekeeper password, and a one-time password generated by the user's CRYPTOCard. After logging in, a prompt will appear. The user must then enter the name of the machine to transfer to (e.g. dave) and enter the unique password for that machine. Passwords on the individual supercomputers will expire every 90 days.

### 2.4.2 Fileserver

The NFS mounted fileserver /fs not longer exists inside the firewall. A new fileserver /fis has been created for use inside the firewall. Please see the web site above for recent changes to this configuration.

### 2.4.3 sftp,scp,ssh

Only the secure versions of ftp, cp and secure shell are allowed. All other commands have been disabled.

### 3 Obtaining code, scripts and input data

this section describes how to download the CCSM3.0 source code and input datasets and the resulting unpacked directory structures.

#### 3.1 Downloading the model

CCSM3.0 is available via the web from:

[www.cgd.ucar.edu/csm/models/CCSM3.0](http://www.cgd.ucar.edu/csm/models/CCSM3.0)

The CCSM3.0 distribution consists of a single tar file, (`ccsm3.0.tar.gz`) that contains the source code and accompanying scripts. In addition there are several input data tar files that need to be selectively downloaded by the user depending on the component set desired as well as the resolution needed.

##### Source code and scripts

<code>ccsm3.0.tar.gz</code>	source code and scripts
-----------------------------	-------------------------

##### Input data

<code>ccsm3.0.inputdata.atm_lnd.tar</code>	CAM/CLM data common to all resolutions
<code>ccsm3.0.inputdata.T85.tar</code>	CAM/CLM T85 specific data
<code>ccsm3.0.inputdata.T42.tar</code>	CAM/CLM T42 specific data
<code>ccsm3.0.inputdata.T31.tar</code>	CAM/CLM T31 specific data
<code>ccsm3.0.inputdata.gx1v3.tar</code>	POP/CSIM gx1 specific data
<code>ccsm3.0.inputdata.gx3v5.tar</code>	POP/CSIM gx3 specific data
<code>ccsm3.0.inputdata.cpl.tar</code>	CPL data
<code>ccsm3.0.inputdata.dxxx.tar</code>	data models
<code>ccsm3.0.inputdata.latm.tar</code>	LATM data
<code>ccsm3.0.inputdata_user.tar</code>	empty directory tree for user-created input datasets
<code>ccsm3.0.inputdata.CLM_RAW_SFC_GEN_FILES.tar</code>	optional files for the generation of CLM3.0 surface datasets at resolutions not included with this release. This is currently not officially supported in the CCSM3.0 release.

#### 3.2 Untarring the source code and scripts

To uncompress and untar the file `ccsm3.0.tar.gz`, use the Unix **gunzip** and **tar** commands (note that `>` below indicates a Unix prompt):

```
> gunzip -c ccsm3.0.tar.gz | tar -xf -
```

Once untarred, the resulting directory tree is:

```

$CCSMROOT/
|
ccsm3/
|
+-----+
|               |
scripts/        models/
|               |
[build/run scripts] [model code]
|               |
+-----+
|               |               |
ccsm_utils/    create_newcase*  create_test*
|
+-----+-----+-----+-----+-----+-----+-----+-----+...
|               |               |               |               |               |               |
cpl/            atm/            ocn/            ice/            lnd/            bld/            utils/        dead/
|               |               |               |               |               |               |               |
|               +-----+-----+               +-----+               +-----+...   |
|               |               |               |               |               |               |               |
cpl6/   cam/   datm6/   latm6/   pop/   docn6/   csim4/   dice6/   clm2/   dlnd6/   mct/   esmf/   dead6/

```

Note that not all of the directory names (e.g. `clm2/` and `csim4`) have kept up with the official released versions of these models (e.g. `CLM3` and `CSIM5`).

The code distribution consists of two top-level directories:

```

$CCSMROOT/
|
ccsm3/
|
+-----+-----+
|               |
scripts/        models/

```

In the `scripts/` directory contains the scripts and tools required to construct the supported CCSM configurations. The `models/` directory contains all the source code needed to build the CCSM. Underneath the `models/` directory are directories for each of the CCSM components:

<code>models/bld/</code>	Gnumake Makefiles and machine-dependent macro files
<code>models/cpl</code>	Source code for the flux coupler
<code>models/atm/</code>	Source code for the atmosphere models
<code>models/ice/</code>	Source code for the sea-ice models
<code>models/ocn/</code>	Source code for the ocean models
<code>models/lnd/</code>	Source code for the land models
<code>models/utils/</code>	Source code for common utilities (e.g. <code>esmf</code> , <code>mct</code> )
<code>models/csm_share/</code>	Source code shared by multiple CCSM components (e.g. <code>shr/</code> <code>cpl/</code> )

### 3.3 Untarring the input data

The various downloaded input data tar files should be untarred into a common **inputdata** directory. In addition, a CCSM resolution must first be selected before the inputdata is downloaded. For example, if a user wishes to run a T42\_gx1v3 CCSM3 run with fully active components, they should untar the input data as follows (we assume the user will be untarring the data into `/user/data`):

```
> cd /user/data
> tar -xf ccsm3.0.inputdata.T42_gx1v3.tar
> tar -xf ccsm3.0.inputdata.gx1v3.tar
> tar -xf ccsm3.0.inputdata.atm\_lnd.tar
> tar -xf ccsm3.0.inputdata.cpl.tar
```

This will result in an `inputdata/` directory tree as shown in section 3.3 but only containing sub-directories for `cpl/`, `cam2/`, `pop/`, `csim4/` and `clm2/`.

The directory tree for the tarfile containing the CCSM3.0 input data is shown below.

```

$DIN_LOC_ROOT/
|
inputdata/
|
+-----+
|       |       |       |       |
cpl/    atm/    ocn/    ice/    lnd/
|       |       |       |       |
+-----+ +-----+ +-----+ +-----+
|       |       |       |       |       |
cam2/  datm6/  latm6/  pop/   docn6/  csim4/  dice6/  clm2/  dlnd6/

```

Note that there is one input data directory for every model. Each model directory in turn contains sub-directories containing input data for particular model component. In addition, some of above component directories also contain subdirectories of data. These are listed below:

Table 4: `inputdata/` sub-directories

Directory	Description
<code>atm/cam2/inic/gaus</code>	cam initial files for spectral dycores
<code>atm/cam2/inic/fv</code>	cam initial files for finite-volume dycore
<code>atm/cam2/rad</code>	cam solar constant ramps, absorptivity/emissivity, aerosol optical properties and prescribed aerosol distributions
<code>atm/cam2/ozone</code>	cam prescribed ozone distributions
<code>atm/cam2/ggas/</code>	cam greenhouse gas chemistry and ramping datasets
<code>atm/cam2/scyc/</code>	cam sulfur cycle chemistry datasets
<code>atm/datm6/</code>	datm cycling data
<code>atm/latm6/</code>	latm cycling data
<code>ocn/pop/gx1v3/ic/</code>	pop initial conditions datasets
<code>ocn/pop/gx3v5/ic/</code>	pop initial conditions datasets
<code>ocn/pop/gx1v3/forcing/</code>	pop forcing datasets
<code>ocn/pop/gx3v5/forcing/</code>	pop forcing datasets
<code>ocn/pop/gx1v3/grid/</code>	pop grid-related input datasets
<code>ocn/pop/gx3v5/grid/</code>	pop grid-related input datasets

Table 4: inputdata/ sub-directories

ocn/docn6/	docn cycling data
ice/csim4/	csim data
ice/dice6/	dice cycling data
lnd/clm2/inidata.2.1/ccsm	clm initial data
lnd/clm2/srfddata/csm	clm surface data
lnd/clm2/rtmdata	clm river-routing (RTM) data
lnd/clm2/pftdata	clm pft-physiology data
lnd/clm2/rawdata	clm raw data to create surface datasets at model resolution
lnd/dlnd6	dlnd cycling data
cpl/cpl6/	coupler data

The initial and boundary datasets for each component will be found in their respective subdirectories. Because it could take up several Gigabytes of disk space depending upon the resolution of the input datasets downloaded, the input data tar files should be untarred on a disk with sufficient space.

Along with the released input datasets, we are also providing an empty directory tree, `inputdata_user/`, that has the same structure as `inputdata/`, but without any files. This tree is provided for user-created input datasets. It should be untarred in a directory parallel to the `inputdata/` directory. It should not be untarred under `inputdata/`. If the user intends to replace an existing `inputdata/` dataset with their own customized dataset, the new dataset should be give a **unique** filename and placed in the corresponding directory as the original `inputdata/` file that it replaces. More details are provided in section 6.12.

### 3.4 Testing the downloaded source code and data files

After downloading the model source code and data sets, users are **strongly advised** to validate the download by running low-resolution tests described in section 7.3.1. T31 and gx3v5 data sets are required to run the recommended test suite.

## 4 Creating cases (preparing model runs)

CCSM3 cases refer to the way a user creates a particular CCSM model run. It refers to both the “name” of the model run (and the name associated with model run output data) as well as the directory name where the scripts for the model run will be generated.

In what follows and throughout this document, we will use the following naming and style convention:

- **xxx** - executable scripts and associated options are in bold helvetica
- xxx - files and directories are in helvetica
- *\$xxx* - environment variables are in italics
- *\$CASE* - defines both case name and case directory name containing build and run scripts
- *\$MACH* - supported machine name
- *\$CASEROOT/* - root directory for case scripts (e.g. *\$HOME/\$CASE/*)
- *\$CCSMROOT/* - root directory for ccsm source code and scripts

### 4.1 Scripts usage overview

The CCSM3.0 scripts have been significantly upgraded from those in CCSM2. The new scripts are based on a very different design philosophy, making their use much easier for both novice as well as expert users. The new scripts operate by generating *resolved* scripts (see section 4.3) for a specific component set, resolution and CCSM3 initialization type. In addition, machine-specific build, run and archiving scripts are produced for each machine requested. The script generation process does not have to occur on the production machine and can occur on a front end machine if needed. Only the user-generated *resolved* build and run scripts have to be executed on the production machine. The README file in the *ccsm3/scripts/* directory provides up-to-date information on how to use the CCSM3.0 scripts. Users should reference that file for additional information.

The following steps provide a summary of the steps necessary to create, build and run a model case using the CCSM3.0 scripts. These steps will be covered in detail in sections 4.2, 4.3, 4.4.1, 4.6, 4.7, 4.8 and 6.

1. `cd $CCSMROOT/ccsm3/scripts/`
2. `invoke ./create_newcase -case $CASEROOT -mach $MACH` (see section 4.2)
3. `cd $CASEROOT/`
4. optionally edit `env_conf` and `tasks/threads` portion of `env_mach.$MACH`
5. `invoke configure -mach $MACH`
6. optionally edit `env_run` and `non-tasks/threads` portion of `env_mach.$MACH`
7. interactively run `./$CASE.$MACH.build` (see section 4.4.1)
8. submit **`$CASE.$MACH.run`** to the batch queue on *\$MACH* (see section 4.4.1)

As shown above, the user must invoke the two commands, **create\_newcase** and **configure** to generate a new model case. Invoking **create\_newcase** (as shown below) results in the generation of a new *\$CASEROOT/* directory containing environment variable files and the script **configure**. Executing **configure** results in the creation of several *\$CASEROOT/* subdirectories containing scripts for namelist generation, input data prestaging, and creation of component executables and required CCSM3 libraries. *\$CASEROOT/* also contains scripts for building, running, and performing long- term data archiving on machine *\$MACH*.

## 4.2 create\_newcase

The script, **create\_newcase**, is the starting point for the generation of a new CCSM3 case. The **create\_newcase** script exists in the `$CCSMROOT/ccsm3/scripts/` directory and **must be run from this directory**. Invoking **create\_newcase** generates environment variable files and an accompanying **configure** script in a new user-specified case directory.

To obtain a concise usage summary type:

```
> create_newcase
```

To obtain a detailed usage summary type:

```
> create_newcase -help
```

To obtain a new case directory type:

```
> create_newcase -case $CASEROOT -mach $MACH
    [-res resolution] [-compset <component set name>]
    [-ccsmroot <ccsm root directory>]
```

The **-case** and **-mach** arguments are required. If `$CASEROOT` does not contain a full pathname (e.g. it is simply set to `$CASE`), a new case directory (`$CASEROOT/`) will be generated in the `$CCSMROOT/ccsm3/scripts/` directory. This is not recommended and the user is urged to set `$CASEROOT` to a full pathname. Although a target machine must be specified on the **create\_newcase** command line, other machines can be added to `$CASEROOT/` at a later time (see section 6.10). Typing

```
> ./create_newcase -case /home/user/$CASE -mach $MACH
```

will result in the creation of a new directory `/home/user/$CASE` (i.e. `$CASEROOT`) which will in turn contain the following:

- **configure**
- `env.readme`
- `env.conf`
- `env_run`
- `env_mach.$MACH`
- `SourceMods/`

The files `env.conf`, `env_run` and `env_mach.$MACH` contain environment variables that define the CCSM run. The `SourceMods/` subdirectory is an optional location for user-modified source code. Optional arguments to **create\_newcase** are resolution (**-res**), component set name (**-compset**), and the CCSM3 root directory (**-ccsmroot**). Including the arguments **-res** and **-compset** will modify the default `env.conf` file that is generated. If no optional arguments provided to **configure**, the resolution will be set to `T42_gx1v3` and the component set will be set to `B` in `env.conf`. As an example, running

```
> ./create_newcase -case /user/case1 -mach bluesky -res T85_gx1v3 -compset K
```

will result in a new case directory, (`/user/case1/`) containing an `env.conf` file which will set up a CCSM run using component set `K` (`cam`, `clm`, `dice` and `docn`, `cpl`, see section 1.3.1) at `T85_gx1v3` resolution (see section 1.3.2). The directory, (`/user/case1/`), will also contain an `env_mach.bluesky` file.

### 4.3 Resolved scripts

The concept of “resolved” scripts is new to CCSM3. A “resolved” namelist-prestaging script generates a component namelist and prestages component input data for a given CCSM resolution and a given way of starting up the CCSM run. Different initialization datasets are required for different types of CCSM initializations (see 5.4). Similarly, different initialization datasets are required for different CCSM resolutions. For instance, input datasets needed for a T42\_gx1v3 run are different than those needed for a T31\_gx3v5 run. Components comprising a startup T42\_gx1v3 run will have different namelists from those comprising a branch T42\_gx1v3 run. Components comprising a startup T31\_gx1v5 run will have different namelists from those comprising a startup T42\_gx1v3 run. Finally, component set B at T42\_gx1v3 will have different namelists than component set A at T42\_gx1v3.

### 4.4 configure

The script **configure** generates “resolved” CCSM3 scripts in the `$CASEROOT/` directory. These will function to create component namelists, prestage the necessary component input data and build and run the CCSM model on a target machine. **configure must be run from the \$CASE directory.**

The environment variables listed in `env_conf` determine what is “resolved” when **configure** is invoked and consequently what may not be modified once the resolved scripts exist. In particular, the contents of the scripts generated by **configure** (in `Buildnml_prestage/` and `Buildexe/`) depend on the values of the environment variables in `env_conf`. In general, `env_conf` resolves model resolution, model component sets and model initialization type (e.g. startup, branch or hybrid). Consequently, `env_conf` must be modified before **configure** is run. Once **configure** is invoked `env_conf` may be modified only by running **configure -cleannall** (see below) first.

In addition, running **configure** for a given machine also generates batch queue commands for the given machine that depend on the task/thread settings in `env_mach.$MACH`. Consequently, if values other than the default values listed are desired, these should be changed before running **configure**. In this case, the following lines in `env_mach.$MACH` must be modified before running **configure**):

```
setenv NTASKS_ATM $ntasks_atm
setenv NTHRDS_ATM $nthrds_atm
setenv NTASKS_LND $ntasks_lnd
setenv NTHRDS_LND $nthrds_lnd
setenv NTASKS_ICE $ntasks_ice
setenv NTHRDS_ICE $nthrds_ice
setenv NTASKS_OCN $ntasks_ocn
setenv NTHRDS_OCN $nthrds_ocn
setenv NTASKS_CPL $ntasks_cpl
setenv NTHRDS_CPL $nthrds_cpl
```

The usage for **configure** is as follows.

To obtain a concise summary of the usage type:

```
> configure
```

To obtain extensive documentation of the usage type:

```
> configure -help
```

To invoke the various configure options type:

```
> configure [-mach $MACH] [-addmach $MACH] [-cleanmach $MACH] [-cleanall]
```

A full summary of each option is provided below.

#### 4.4.1 **configure -mach**

Running **configure -mach \$MACH** creates the following sub-directories and files in the **\$CASE-ROOT/** directory:

- **.cache/**  
Contains files that enable the scripts to perform validation tests when building or running the model.
- **Buidexe/**  
Contains “*resolved*” scripts to generate model executables for each model component in the requested CCSM3 component set.
- **Buildlib/**  
Contains scripts to generate required built-in CCSM3 libraries (e.g. ESMF, MCT, etc.).
- **Buildnml\_Prestage/**  
Contains “*resolved*” scripts to generate component namelists and prestage component input data.
- **\$CASE.\$MACH.build**  
Creates the executables necessary to run CCSM3 (see section 5.2)
- **\$CASE.\$MACH.run**  
Runs the CCSM3 model and performs short term archiving (see section 5.5.1).
- **\$CASE.\$MACH.l\_archive**  
Performs long-term archiving (see section 5.5.2).

#### 4.4.2 **configure -cleanmach and configure -cleanall**

The **configure** options **-cleanmach** and **-cleanall** provide the user with the ability to make changes to **env\_conf** or **env\_mach.\$MACH** after **configure -mach \$MACH** has been invoked. The **configure** script will stop with an error message if a user attempts to recreate scripts that already exist. The build and run scripts will also recognize changes in **env\_conf** or **env\_mach.\$MACH** that require **configure** to be rerun. Note that the options **-cleanall** and **-cleanmach** are fundamentally different.

Running **configure -cleanmach \$MACH** renames the build, run, and l\_archive scripts for the particular machine and allows the user to reset the machine tasks and threads and rerun **configure**. It is important to note that the **Build\*/** directories will NOT be updated in this process. As a result, local changes to namelist, input data, or the environment variable files will be preserved.

Running **configure -cleanall** removes all files associated with all previous invocations of the **configure** script. The **\$CASEROOT/** directory will now appear as if **create\_newcase** had just been run with the exception that local modifications to the **env\_\*** files are preserved. All **Build\*/** directories will be removed, however. As a result, any changes to the namelist generation scripts in **Buildnml\_prestage/** will NOT be preserved. Before invoking this command, users should make backup copies of their “*resolved*” component namelists in the **Buildnml\_Prestage/** directory if modifications to the generated scripts were made.

#### 4.4.3 **configure -addmach**

The advantage of having scripts for more than one machine in one **\$CASEROOT/** directory is that the same “*resolved*” scripts in **Buildnml\_prestage/** and **Buidexe/** can be used on multiple machines. Since long production runs are often performed on more than one machine during the course of a model run, this mechanism makes it very seamless to change machines during the course of a single model run.

Running **configure -addmach \$MACH** allows the user to add new machines to an already generated **\$CASEROOT/** directory. For instance, if a **\$CASEROOT/** directory and accompanying scripts were created for bluesky, the following build and run scripts for bluesky would exist in the **\$CASEROOT/** directory:

- **\$CASE.bluesky.build**

- **`$CASE.bluesky.l_archive`**
- **`$CASE.bluesky.run`**

Users can generate scripts for blackforest in the `$CASEROOT/` directory as follows:

1. `cd $CASEROOT/`
2. `./configure -addmach blackforest` (this adds `env_mach.blackforest` to `$CASEROOT/`)
3. optionally edit `env_mach.blackforest` to change default values of tasks and/or threads
4. `./configure -mach blackforest`

The following additional scripts are then produced in the `$CASEROOT/` directory:

- **`$CASE.blackforest.build`**
- **`$CASE.blackforest.l_archive`**
- **`$CASE.blackforest.run`**

## 4.5 Environment variables

The files `env_conf`, `env_run` and `env_mach.$MACH` contain environment variables needed to generate “resolved” scripts, build and run the CCSM model and perform short and long-term archiving on output data. The following table give a summary of the environment variables in each file. Those entries where “User Modification” is “yes” denote environment variables that the user might want to modify whereas those where “User Modification” is “no” are environment variable that are given reasonable default values and that in general the user might not want to change. A full discussion of each of these environment variables is given in sections 4.6, 4.7 and 4.8. The same summary can be found in `$CASEROOT/env.readme`.

Table 5: `env_conf`, `env_run` and `env_mach.$MACH` variables

File	Environment Variable(s)	User Modification
<code>env_conf</code>	<code>\$CASE</code>	yes
<code>env_conf</code>	<code>\$CASEST</code>	yes
<code>env_conf</code>	<code>\$COMP_ATM</code> , <code>\$COMP_LND</code> , <code>\$COMP_ICE</code> <code>\$COMP_OCN</code> , <code>\$COMP_CPL</code>	yes
<code>env_conf</code>	<code>\$CSIM_MODE</code>	yes
<code>env_conf</code>	<code>\$GRID</code>	yes
<code>env_conf</code>	<code>\$RUN_TYPE</code>	yes
<code>env_conf</code>	<code>\$RUN_STARTDATE</code>	yes
<code>env_conf</code>	<code>\$RUN_REFCASE</code> , <code>\$RUN_REFDATE</code>	yes
<code>env_conf</code>	<code>\$IPCC_MODE</code>	yes
<code>env_conf</code>	<code>\$RAMP_CO2_START_YMD</code>	yes
<code>env_run</code>	<code>\$RESUBMIT</code>	yes
<code>env_run</code>	<code>\$CCSMROOT</code>	yes
<code>env_run</code>	<code>\$CASEROOT</code>	yes
<code>env_run</code>	<code>\$CONTINUE_RUN</code>	yes
<code>env_run</code>	<code>\$STOP_OPTION</code> , <code>\$STOP_N</code>	yes
<code>env_run</code>	<code>\$REST_OPTION</code>	no
<code>env_run</code>	<code>\$REST_N</code>	no
<code>env_run</code>	<code>\$INFO_DEBUG</code>	no
<code>env_run</code>	<code>\$DEBUG</code>	no

Table 5: `env_conf`, `env_run` and `env_mach.$MACH` variables

<code>env_run</code>	<code>\$SETBLD</code>	no
<code>env_run</code>	<code>\$OCN_TRACER_MODULES</code>	no
<code>env_run</code>	<code>\$HIST_OPTION</code>	no
<code>env_run</code>	<code>\$HIST_N</code>	no
<code>env_run</code>	<code>\$HIST_64BIT</code>	no
<code>env_run</code>	<code>\$AVHIST_OPTION</code>	no
<code>env_run</code>	<code>\$AVHIST_N</code>	no
<code>env_run</code>	<code>\$DIAG_OPTION</code>	no
<code>env_run</code>	<code>\$DIAG_N</code>	no
<code>env_run</code>	<code>\$LOGDIR</code>	no
<code>env_mach.\$MACH</code>	<code>\$NTASKS_ATM,\$NTHRDS_ATM, \$NTASKS_LND,\$NTHRDS_LND, \$NTASKS_ICE,\$NTHRDS_ICE, \$NTASKS_OCN,\$NTHRDS_OCN, \$NTASKS_CPL,\$NTHRDS_CPL</code>	yes
<code>env_mach.\$MACH</code>	<code>\$MAX_TASKS_PER_NODE (IBM only)</code>	no
<code>env_mach.\$MACH</code>	<code>\$EXEROOT</code>	yes
<code>env_mach.\$MACH</code>	<code>\$RUNROOT</code>	no
<code>env_mach.\$MACH</code>	<code>\$GMAKE_J</code>	yes
<code>env_mach.\$MACH</code>	<code>\$DIN_LOC_ROOT, DIN_LOC_ROOT_USER, \$DIN_LOC_MSROOT, \$DIN_REM_MACH, DIN_REM_MSROOT, \$DIN_REM_ROOT</code>	yes
<code>env_mach.\$MACH</code>	<code>\$DOUT_S, DOUT_S_ROOT</code>	yes
<code>env_mach.\$MACH</code>	<code>\$DOUT_L_MS \$DOUT_L_MSNAME, DOUT_L_MSROOT \$DOUT_L_MSPWD, DOUT_L_MSRPD, \$DOUT_L_MSPRJ \$DOUT_L_RCP, DOUT_L_RCP_ROOT</code>	yes

#### 4.6 Environment variables in `env_conf`

**name:** **CASE**  
**description:** Case name (short identifier for run)  
**valid values:** string of up to 80 characters

**name:** **CASESTR**  
**description:** Descriptive Case String (identifier for run)  
**valid values:** string of up to 256 characters

**name:** **COMP\_ATM**  
**description:** Atmospheric component name. Component build and namelist-prestage generation scripts will only be created for this specified component name  
**valid values:** cam, datm, latm or xatm  
 Default setting is cam.

<b>name:</b>	<b>COMP_LND</b>
<b>description:</b>	Land component name. Component build and namelist-prestage generation scripts will only be created for this specified component name
<b>valid values:</b>	clm, dlnd or xlnd Default setting is clm
<b>name:</b>	<b>COMP_ICE</b>
<b>description:</b>	Ice component name. Component build and namelist-prestage generation scripts will only be created for this specified component name
<b>valid values:</b>	csim, dice or xice Default setting is csim
<b>name:</b>	<b>COMP_OCN</b>
<b>description:</b>	Ocean component name. Component build and namelist-prestage generation scripts will only be created for this specified component name
<b>valid values:</b>	pop, docn or xocn Default setting is pop
<b>name:</b>	<b>COMP_CPL</b>
<b>description:</b>	Coupler component name. Component build and namelist-prestage generation scripts will only be created for this specified component name
<b>valid values:</b>	cpl
<b>name:</b>	<b>CSIM_MODE</b>
<b>description:</b>	CSIM model specification
<b>valid values:</b>	prognostic or ocean_mixed_ice A setting of prognostic implies that the complete CSIM ice model will be used. A setting of ocean_mixed_ice implies that the slab ocean mixed layer within CSIM is utilized. See the Community Sea Ice Model (CSIM) User's Guide Version 5.0 for more details. Default setting is prognostic
<b>name:</b>	<b>GRID</b>
<b>description:</b>	CCSM grid resolution
<b>valid values:</b>	T85_gx1v3, T42_gx1v3, T31_gx3v5, 2x2.5_gx1v3, T62_gx1v3 or T62_gx3v5 Default setting is T42_gx1v3
<b>name:</b>	<b>RUN_TYPE</b>
<b>description:</b>	Type of CCSM initialization
<b>valid values:</b>	startup, branch or hybrid (see section 5.4) Default setting is startup

**name:** **RUN\_STARTDATE**  
**description:** Start date of CCSM run. Only used for startup or hybrid runs. Ignored for branch runs (see section 5.4).  
**valid values:** YYYY-MM-DD (e.g. 1990-01-01)

**name:** **RUN\_REFCASE**  
**description:** Reference case for branch or hybrid runs. Ignored for startup runs (see section 5.4).  
**valid values:** string of up to 80 characters

**name:** **RUN\_REFDATE**  
**description:** Reference date for branch or hybrid runs. Ignored for startup runs (see section 5.4).  
**valid values:** YYYY-MM-DD (e.g. 1990-01-01)

**name:** **IPCC\_MODE**  
**description:** Turns on various supported IPCC mode configurations  
**valid values:** OFF, 1870\_CONTROL, RAMP\_CO2\_ONLY  
 By default, *\$IPCC\_MODE* is set to OFF.

**name:** **RAMP\_CO2\_START\_YMD**  
**description:** Start date of CAM CO2 ramp  
 This **MUST** be set if *IPCC\_MODE* is set to RAMP\_CO2\_ONLY.  
 This variable is ignored if *\$IPCC\_MODE* is not set to RAMP\_CO2\_ONLY.  
**valid values:** YYYYMMDD

#### 4.7 Environment variables in `env_run`

**name:** **RESUBMIT**  
**description:** Flag to determine if the model should resubmit itself at the end of a run. If *\$RESUBMIT* is 0, then the run script will not resubmit itself. If *\$RESUBMIT* is greater than 0, then the case run script will resubmit itself, decrement *\$RESUBMIT* by 1 and set the value of *\$CONTINUE\_RUN* to TRUE.  
**valid values:** an integer greater than or equal to 0

**name:** **CASEROOT**  
**description:** Root model case directory (full pathname containing where *\$CASE* directory and also containing the files *env\_conf*, *env\_run*, etc.)

**name:** **CCSMROOT**  
**description:** Root ccsm source directory (contains the directories *models/*, *scripts/*, etc.)

<b>name:</b>	<b>CONTINUE_RUN</b>
<b>description:</b>	Flag to specify whether run is a continuation run. A continue run extends an <b>existing</b> CCSM run exactly, where "exactly" means that exact same answers at the bit-level are obtained as if the existing run had not been stopped. A run may be continued indefinitely from an existing run.
<b>valid values:</b>	TRUE or FALSE
<b>name:</b>	<b>OCN_TRACER_MODULES</b>
<b>description:</b>	POP model passive tracer specification. Used only by POP scripts. Setting the passive tracer modules to ( ) results in only 2 tracers (the default in POP).
<b>valid values:</b>	(iage), (cfc), (iage cfc), or ( ) Default setting is (iage)
<b>name:</b>	<b>DEBUG</b>
<b>description:</b>	Flag to turn on run and compile time debugging.
<b>valid values:</b>	TRUE or FALSE Default setting is FALSE
<b>name:</b>	<b>SETBLD</b>
<b>description:</b>	Flag to specify whether model will be built at runtime.
<b>valid values:</b>	AUTO, TRUE or FALSE Models use the <i>\$BLDTYPE</i> environment variable, which is determined from the setting of <i>\$SETBLD</i> , to determine if a build should be executed at run time. Each component build script checks this flag to determine if gmake is to be invoked. If <i>\$SETBLD</i> is TRUE, gmake will always be invoked on each component at run time. If <i>\$SETBLD</i> is FALSE, gmake will not be invoked on each component at run time. If <i>\$SETBLD</i> is AUTO, gmake will only be invoked on each component if <i>\$CONTINUE_RUN</i> is FALSE. Default setting is TRUE
<b>name:</b>	<b>STOP_OPTION</b>
<b>description:</b>	Coupler namelist variable that sets the ending simulation time. If <i>\$STOP_OPTION</i> is set to: date - model stops on given date. ndays - model stops every <i>\$STOP_N</i> days relative to start date nmonths - model stops every <i>\$STOP_N</i> months relative to start date. daily - model stops every day monthly - model stops every 1st day of month yearly - model stops every 1st day of year
<b>valid values:</b>	date, ndays, nmonths, daily, monthly, or yearly Default setting is ndays.

<b>name:</b>	<b>STOP_N</b>
<b>description:</b>	Coupler namelist variable that provides additional information with respect to <i>\$STOP_OPTION</i> . If <i>\$STOP_OPTION</i> is set to: ndays - <i>\$STOP_N</i> is the number of days to run. nmonths - <i>\$STOP_N</i> is the number of months to run. date - daily, monthly, or yearly, this option is ignored.
<b>valid values:</b>	integer Default setting 5
<b>name:</b>	<b>REST_OPTION</b>
<b>description:</b>	Coupler namelist variable that sets the frequency at which restart files are created. If <i>\$REST_OPTION</i> is set to: ndays - restart files are generated every <i>\$REST_N</i> days relative to start date nmonths - restart files are generated every <i>\$REST_N</i> months relative to start date daily - restart files are generated every day monthly - restart files are generated every first day of every month yearly - restart files are generated every first day of every year
<b>valid values:</b>	date, ndays, nmonths, daily, monthly, or yearly Default setting is <i>\$STOP_OPTION</i> .
<b>name:</b>	<b>REST_N</b>
<b>description:</b>	Coupler namelist variable that provides additional information with respect to <i>\$REST_OPTION</i> . If <i>\$REST_OPTION</i> is set to: ndays - restart files are generated every <i>\$REST_N</i> days. nmonths - restart files are generated every <i>\$REST_N</i> months. date - daily, monthly, or yearly, this option is ignored.
<b>valid values:</b>	integer Default setting is <i>\$STOP_N</i> .
<b>name:</b>	<b>HIST_OPTION</b>
<b>description:</b>	Coupler namelist variable that sets the frequency that coupler history files are created. This does not apply to other model component history files. If <i>\$HIST_OPTION</i> is set to: date - coupler history files are generated on given date ndays - coupler history files are generated every <i>\$HIST_N</i> days relative to start date nmonths - coupler history files are generated every <i>\$HIST_N</i> months relative to start date daily - coupler history files are generated every day monthly - coupler history files are generated every first day of every month yearly - coupler history files are generated every first day of every year never - coupler history files are never
<b>valid values:</b>	date, ndays, nmonths, daily, monthly, yearly or never Default setting is never.

**name:** **HIST\_N**  
**description:** Coupler namelist variable that provides additional information with respect to *\$HIST\_OPTION*.  
 If *\$HIST\_OPTION* is set to:  
 ndays - coupler history files are generated every *\$HIST\_N* days.  
 nmonths - coupler history files are generated every *\$HIST\_N months*.  
 date - daily, monthly, or yearly, this option is ignored.  
**valid values:** integer  
 Default settings is -999.

**name:** **HIST\_DATE**  
**description:** Coupler namelist variable that provides additional information with respect to *\$HIST\_OPTION*.  
 If *\$HIST\_OPTION* is set to date, *\$HIST\_DATE* is the date at which to write a coupler history file.  
 If *\$HIST\_OPTION* is not set to date, this option is ignored.  
**valid values:** integer

**name:** **DIAG\_OPTION**  
**description:** Coupler namelist variable that sets the frequency that coupler diagnostic files are created.  
 If *\$DIAG\_OPTION* is set to:  
 date - coupler diagnostic files are generated on given date.  
 ndays - coupler diagnostic files are generated every *\$DIAG\_N* days relative to start date  
 nmonths - coupler diagnostic files are generated every *\$DIAG\_N* months relative to start date  
 daily - coupler diagnostic files are generated every day  
 monthly - coupler diagnostic files are generated every first day of every month  
 yearly - coupler diagnostic files are generated every first day of every year  
 never - coupler diagnostic files are never created  
**valid values:** date, ndays, nmonths, daily, monthly, yearly or never  
 Default setting is never.

**name:** **DIAG\_N**  
**description:** Coupler namelist variable that provides additional information with respect to *\$DIAG\_OPTION*.  
 If *\$DIAG\_OPTION* is set to ndays, diagnostic files are generated every *\$DIAG\_N* days.  
 If *\$DIAG\_OPTION* is set to nmonths, diagnostic files are generated every *\$DIAG\_N* months.  
 If *\$DIAG\_OPTION* is set to date, daily, monthly, or yearly, this option is ignored.  
**valid values:** integer  
 Default setting in script is -999.

**name:** INFO\_DEBUG  
**description:** Coupler output information flag. Higher levels result in more information written to every component model's standard output file.  
**valid values:** 0,1,2 or 3  
 Default setting is 1

**name:** LOGDIR  
**description:** Full pathname of directory where stdout and stderr should be copied. If \$LOGDIR is set to "", then stdout and stderr will not be copied to a \$LOGDIR/ directory before short-term archiving occurs.  
**valid values:** full pathname or "" Default setting is ""

## 4.8 Environment variables in env\_mach.\$MACH

### 4.8.1 env\_mach.\$MACH tasks/threads

**name:** NTASKS\_ATM, NTASKS\_LND, NTASKS\_OCN, NTASKS\_ICE, NTASKS\_CPL  
**description:** Number of component MPI tasks for each component.  
**valid values:** integer

**name:** NTHRDS\_ATM, NTHRDS\_LND, NTHRDS\_OCN, NTHRDS\_ICE, NTHRDS\_CPL  
**description:** Number of OpenMP threads per MPI task for each component.  
**valid values:** integer

**name:** MAX\_TASKS\_PER\_NODE  
**description:** Maximum number of MPI tasks per node.  
 Currently, this is only applicable to IBM machines  
**valid values:** integer

### 4.8.2 env\_mach.\$MACH general machine specific settings

**name:** EXEROOT  
**description:** Root executable directory. Model executables are built here.

**name:** RUNROOT  
**description:** Root executable run directory. Model executables are run here.

**name:** GMAKE\_J  
**description:** Number of threads invoked as part of gmake.  
**valid values:** integer greater than 0.

### 4.8.3 env\_mach.\$MACH input data prestaging

<b>name:</b>	<b>DIN_LOC_ROOT</b>
<b>description:</b>	Local disk root directory for officially released CCSM input data. Used by the script <b>Tools/ccsm_getinput</b> .
<b>name:</b>	<b>DIN_LOC_MSROOT</b>
<b>description:</b>	Local mass store root directory for officially released CCSM input data. Used by the script <b>Tools/ccsm_getinput</b> .
<b>name:</b>	<b>DIN_LOC_ROOT_USER</b>
<b>description:</b>	Local disk root directory for user-specific CCSM input data. Used by the script <b>Tools/ccsm_getinput</b> .
<b>name:</b>	<b>DIN_REM_MACH</b>
<b>description:</b>	Name of remote machine where officially released CCSM input data are located.
<b>name:</b>	<b>DIN_REM_ROOT</b>
<b>description:</b>	Remote machine disk root containing officially released CCSM input data.
<b>name:</b>	<b>DIN_REM_MSROOT</b>
<b>description:</b>	Remote machine mass store root containing officially released CCSM input data

### 4.8.4 env\_mach.\$MACH output data short-term archiving environment variables

<b>name:</b>	<b>DOUT_S</b>
<b>description:</b>	Flag to determine if short-term archiving of output data is enabled. A setting of TRUE implies that short-term archiving will be enabled.
<b>valid values:</b>	TRUE or FALSE Default setting is TRUE
<b>name:</b>	<b>DOUT_S_ROOT</b>
<b>description:</b>	Short-term archiving root directory.

### 4.8.5 env\_mach.\$MACH output data long-term archiving

<b>name:</b>	<b>DOUT_L_MS</b>
<b>description:</b>	Flag to determine whether long-term archiving of model output data will be done. <b>long-term archiving of output data is done from the short-term archiving area</b> to a local and possibly a remote mass store using the CCSM long-term archiving script, <b>\$CASE.I.archive</b> . <b>long-term archiving of output data will only be activated only if the environment variable \$DOUT_S is also set to TRUE.</b>

valid values:	TRUE or FALSE Default setting is FALSE
<b>name:</b>	<b>DOUT_L_RCP</b>
description:	Flag to determines if long-term archiving should copy files to another site <b>in addition to copying files to the local mass store</b> . If long-term archiving is required to copy files to another site, this requires that ssh be set up so that passwords are note required
valid values:	TRUE or FALSE Default setting is FALSE
<b>name:</b>	<b>DOUT_L_MSROOT</b>
description:	Local mass store long-term archiving root Used by the script <b>\$CASE.I.archive</b> if available. <b>Ignored if \$DOUT_L_MS is set to FALSE.</b>
<b>name:</b>	<b>DOUT_L_MSNAME</b>
description:	Logname in uppercase. Only used for NCAR MSS. Used by the script <b>\$CASE.I.archive</b> if available. Ignored if <b>\$DOUT_L_MS</b> is set to FALSE.
<b>name:</b>	<b>DOUT_L_PWD</b>
description:	Mass store file write password. Only used for NCAR MSS. Used by script <b>\$CASE.I.archive</b> if available. Ignored if <b>\$DOUT_L_MS</b> is set to FALSE.
<b>name:</b>	<b>DOUT_L_RPD</b>
description:	Mass store file read password. Only used for NCAR MSS. Used by script <b>\$CASE.I.archive</b> if available. Ignored if <b>\$DOUT_L_MS</b> is set to FALSE.
<b>name:</b>	<b>DOUT_L_PRJ</b>
description:	Mass store project charge number Only used for NCAR MSS. Used by script <b>\$CASE.I.archive</b> if available. Ignored if <b>\$DOUT_L_MS</b> is set to FALSE.
<b>name:</b>	<b>DOUT_L_RCP_ROOT</b>
description:	Mass store directory path name (only used for NCAR MSS)
description:	Remote machine directory for to be used if <b>\$DOUT_L_RCP</b> is set to TRUE. Ignored if <b>\$DOUT_L_RCP</b> is set to FALSE.

## 4.9 Scripts design for developers

This section provides a brief overview of the design of the scripts and can be used by CCSM developers to help understand how the scripts are implemented and how they operate.

The highest level scripts, **create\_newcase** and **create\_test**, are in the **\$CCSMROOT/ccsm3/scripts** directory. All supporting utilities are contained in the **\$CCSMROOT/ccsm3/scripts/ccsm\_utils/** directory, which in turn has the following directory structure:



## 4.11 Script generation tools

- **ccsm\_auto.csh -interval seconds -njob #\_of\_jobs**  
Handles resubmission of production jobs on machine moon (Earth Simulator). Could be generalized to other machines if necessary. Checks for jobs on a platform and submits job as needed to keep the machine busy.
- **ccsm\_build.csh**  
Generates portions of the CCSM build script. Called by generate\_batch.csh.
- **generate\_batch.csh**  
High level CCSM script generation tool. Generates case build, run, and l\_archive scripts. Called by **configure**.
- **generate\_resolved.csh**  
High level CCSM script generation tool. Generates “resolved” scripts in Build\* directories in a case. Called by **configure**.
- **generate\_taskthread.csh**  
Echoes task and thread settings. Called by generate\_batch.csh to document tasks and threads in the case run script.
- **restart\_compare.pl file1 file2**  
Perl script that compares two coupler log files for bit-for-bit test. Called by subset of test scripts.
- **taskmaker.pl**  
Perl script that generates task geometries automatically for ibm batch job scripts. Uses tasks and threads set in specific case. Called from batch.ibm.\* scripts.
- **testcase\_setup.csh**  
High level script that runs for all test cases. Modifies env\_\* files for individual test cases. Called by all test case setup scripts.

### 4.11.1 Script validation tools

- **check\_compset [-list] compset**  
Verifies that the argument compset is a valid CCSM component set. Examples of valid component sets are A, B, or X. Returns null if valid or error message if invalid. -list provides a list of valid component sets.
- **check\_machine [-list] machine**  
Verifies that the argument machine is a valid CCSM supported machine. Examples of valid machines are blackforest or jazz. Returns null if valid or error message if invalid. -list provides a list of valid machines.
- **check\_res [-list] resolution**  
Verifies that the argument resolution is a valid CCSM resolution. Examples of valid resolutions are T42\_gx1v3 and T31\_gx3v5. Returns null if valid or error message if invalid. -list provides a list of valid resolutions.
- **check\_testcase [-list] testcase**  
Verifies that the argument resolution is a valid CCSM test case. Examples of valid test cases are ER.01a and DB.01f. Returns null if valid or error message if invalid. -list provides a list of valid test cases.
- **get\_compset model compset**  
Translates model and component set to a specific component name. For instance atm and B returns cam, ocn and B returns pop, ocn and A returns docn. Valid model arguments are atm, lnd, ocn, ice, cpl. Valid compset arguments are any component set available in check\_compset.
- **get\_csimmode model compset**  
Translates model and component set to a specific csim mode. For instance csim and B returns prognostic and csim and M returns oceanmixed.ice. Valid model arguments are csim only at this time. Valid compset arguments are any component set available in check\_compset.

### 4.11.2 Input/output data movement

- **ccsm\_l\_archive.csh**  
Carries out long term archiving. Invoked by `$CASE.$MACH.l_archive`.
- **ccsm\_s\_archive.csh**  
Carries out short term archiving. Invoked by `$CASE.$MACH.run`.
- **ccsm\_cpdata [machine1:][dir1/]file1 [machine2:][dir2/]file2**  
Generic utility to copy a file. Can identify whether cp or scp is required based on an optional machine argument. Takes one or two arguments.
- **ccsm\_getfile [dir1/]file1 [dir2/]file2**  
Hierarchical search for a ccsim input file. Search for that file in several sources including locally, on the NCAR mass store, and on other machines. Copy that file into dir2/file2. Uses the scripts `ccsm_cpdata` and `ccsm_msread`. Takes one or two arguments.
- **ccsm\_getinput [dir1/]file1 [dir2/]file2**  
Hierarchical search for a ccsim input file in the following directories:
  1. current directory
  2. `$DIN.LOC.ROOT/`
  3. `$DIN.LOC.MSROOT/`
 dir1 represents the directory path under `$DIN.LOC.ROOT/` or `$DIN.LOC.MSROOT` to search for the file. Uses the scripts `ccsm_cpdata` and `ccsm_msread`. Takes one or two arguments.
- **ccsm\_getrestart**  
Copies restart files from a specific restart archive directory into the component working directories. This will eventually be extended to look for restart files on the local mass store as well. Takes no arguments.
- **ccsm\_msmkdir mssdir**  
Creates a directory on the local mass store. Takes one argument.
- **ccsm\_msread [mssdir/]file1 [locdir/]file2**  
Copies a file from the local mass store to the local disk. Takes one or two arguments.
- **ccsm\_mswrite [locdir/]file1 mssdir/file2**  
Copies a file to the local mass store from the local disk. Takes two arguments.
- **ccsm\_splitdf [-d]/[-f]/[-h] dir/file**  
Returns the directory path or the filename for an argument like dir/file. Can handle very general cases. -d returns the directory. -f returns the file. -h is help. -f is the default return argument. Takes one argument.

## 5 Building, Running and Input/Output Data

This section presents a general overview of how CCSM3 operates. A full description of how to set up a production run is described in section 6. In what follows we assume that the user has already run **create\_newcase** and **configure** (see section 4).

### 5.1 Model input

CCSM3 input data are provided as part of the release via several input data tar files. The tar files are typically broken down by components and/or resolutions. These files should be downloaded and untarred into a single input data root directory (see section 3.3). Each tar file will place files under a common directory named `inputdata/`. The `inputdata/` directory contains numerous subdirectories and the CCSM3 assume that the directory structure and filenames will be preserved.

The officially released input data root directory name is set in the `env_mach.$MACH` file via the environment variable is `$DIN_LOC_ROOT`. A default setting of `$DIN_LOC_ROOT` is provided for each machine in `env_mach.$MACH`.

The user should edit this value if it does not correspond to their `inputdata/` root. Multiple users can share the same `inputdata/` directory. The files existing in the various subdirectories of `inputdata/` should not have Unix write permission on them.

An empty input data root directory tree is also provided as a future place holder for custom user-generated input datasets. This is set in the `env_mach.$MACH` file via the environment variable `$DIN_LOC_ROOT_USER`. If the user wishes to use any user-modified input datasets in place of the officially released version, these should be placed in the appropriate subdirectory of `$DIN_LOC_ROOT_USER/`.

The appropriate CCSM resolved component scripts (in `$CASEROOT/Buildnml.Prestage/`) must then also be modified to use the new filenames. Any datasets placed in `$DIN_LOC_ROOT_USER/` should have unique names that do not correspond to any datasets in `$DIN_LOC_ROOT/`. The contents of `$DIN_LOC_ROOT/` should not be modified. The user should be careful to preserve these changes, since invoking **configure -cleanall** will remove all user made changes.

### 5.2 Building CCSM3

#### 5.2.1 Building summary

CCSM3 can be built by either interactively running **\$CASE.\$MACH.build** or by batch submission of **\$CASE.\$MACH.run** (since **\$CASE.\$MACH.build** is executed automatically from **\$CASE.\$MACH.run**). We recommend that CCSM3 be built interactively. There are several reasons for this. First, building interactively allows the user to immediately detect build related problems without waiting in the batch queueing system. Second, the build process normally occurs on a much smaller set of processors than is used to run CCSM. Consequently, an interactive build saves computing cycles.

The **\$CASE.\$MACH.build** script does the following:

- sources `env_conf`, `env_run` and `env_mach.$MACH`
- creates both the short-term archiving directory (`$DOUT_S_ROOT/`) and the necessary model executable directory hierarchy `$EXEROOT`
- executes **ccsm\_getrestart**
- performs a validity check by examining files in `.cache/` and comparing them with the requested configuration
- checks whether a CCSM build is required
- creates the necessary CCSM libraries
- creates the model executable directories
- creates component namelist files and prestages input data

- creates component executables
- sets up other required input files

Input data prestaging is carried out as part of the build procedure via calls to **ccsm\_getinput**, **ccsm\_getfile** and **ccsm\_getrestart**. These scripts reside in `$CCSMROOT/ccsm3/scripts/ccsm_utils/Tools/`. The script, **\$CASE.\$MACH.build**, always calls **ccsm\_getrestart**, which attempts to copy each component's restart files and associated restart pointer file from the directory, `$DOOUT_S_ROOT/restart/`, to the component's executable directory. If the copy is not successful, a warning message will be printed and the **\$CASE.\$MACH.build** script will continue. We note that successfully obtaining restart files using **ccsm\_getrestart** depends on the activation of short-term archiving (see section 5.5 in order to populate the short term archive restart directory. We also note that a CCSM3 restart run is produced by setting the environment variable `$CONTINUE_RUN` to TRUE in `env_run`. If `$CONTINUE_RUN` is set to TRUE, each component's restart files and associated restart pointer file must be either in the directory `$DOOUT_S_ROOT/restart/`, that component's executable directory or available from the long-term archiving area.

If the build has occurred successfully, the user will see the following message:

```
-----
- CCSM BUILD HAS FINISHED SUCCESSFULLY
-----
```

If this message is not seen, a compiler problem has probably occurred. The user should carefully review the build output to determine the source of the problem

### 5.2.2 Gmake details

Each CCSM component generates its executable by invoking **gmake**. The makefile and corresponding machine-specific makefile macros are found in the directory `$CCSMROOT/ccsm3/models/bld/`:

```

$CCSMROOT/ccsm3/models/bld/
|
+-----+
|       |       |
| makdep.c   Makefile   Macros.*

```

- makdep.c evaluates the code dependencies for all files in all the directories contained in Filepath.
- Makefile - generic gnumakefile for all the models.
- Macros.AIX - build settings specific to AIX (IBM SP2) platforms.
- Macros.ESOS - build settings specific to the Earth Simulator platforms
- Macros.IRIX64 - build settings specific to IRIX64 (SGI Origin) platforms
- Macros.Linux - build settings specific to Linux platforms.
- Macros.OSF1 - build settings specific to OSF1 (Compaq) platforms
- Macros.SUPER-UX - build settings specific to NEC SX6 platform
- Macros.UNICOS - build settings specific to Cray X1 platforms

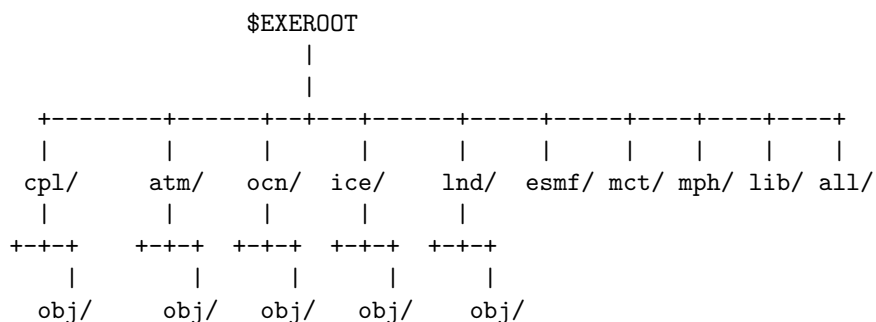
The Macros.\* files contain machine-specific makefile directives. In the current release, the Macros have been divided into different platform-dependent files each containing site/machine specific options. The site and operating system characteristics are set in the machine-specific file, `env_mach.$MACH` via the environment variables `$OS` and `$SITE`. In addition, machine dependent options are also included in the Macros files for specific platforms that have been tested. The machine-specific options are set in the Macros files by use of the environment variable `$MACH`.

If a user needs to modify compiler options for a specific machine, only the machine specific Macros files needs to be edited. Similarly, if a user wants to add a new machine to the CCSM3 scripts, they will need to add or modify the Macros files appropriately to support that new machine. More information about porting CCSM3 to a new machine is available in section 6.10.

For most CCSM components, the specific files used to build each CCSM component are defined in a file called “Filepath”. This file is generated by the scripts, **\$CASEROOT/Buildexe/\*.buildexe.csh**, and contains a list of directories specifying the search path for component source code. The directories listed in Filepath appear in order of importance, from most important first to least important last. If a piece of code appears in two of the listed directories, the code in the directory appearing first will be used and all other versions will be ignored. No error is generated if a directory listed in Filepath does not exist or does not contain any code.

The CCSM3 make system generates file dependencies automatically. Users do not need to maintain these dependencies manually. The makdep.c code is compiled and run by most components prior to building.

The CCSM3 model is built in the directory specified by **\$EXEROOT** (in `env_mach.$MACH`). **\$EXEROOT/** contains a set of subdirectories where each component executable will be built.



Each subdirectory in **\$EXEROOT/** contains the component executable, input datasets, and namelist needed to run that specific CCSM component. For each component, the **\$obj/** directory contains all files created during the compilation of that component. This includes the dependency files, cpp products and object files. Component output data, such as standard out logs, history and restart datasets will also be written into that component’s **\$EXEROOT/** subdirectory. Some of the components, such as POP and CSIM, have separate subdirectories for input, restart and history data, while CAM and CLM output all of these into one directory.

### 5.2.3 User modified source code

Each component **\*.buildexe.csh** script has a directory, **\$CASEROOT/SourceMods/src.xxx/** (where xxx is the component name, e.g. cam) as the first Filepath directory. This allows user modified code to be easily introduced into the model by placing the modified code into the appropriate **\$CASEROOT/SourceMods/src.xxx/** directory.

## 5.3 Running CCSM3

The CCSM3 run script, **\$CASE.\$MACH.run**, is generated as a result of invoking **configure** and is normally submitted in batch mode after the model executables have been built interactively. The specific command required to submit this script is machine dependent. Common batch submission commands in the scripts are “`llsubmit $CASE.$MACH.run`” and “`qsub $CASE.$MACH.run`”. It is worthwhile to note that CCSM can be run interactively *if* there are appropriate resources available.

Upon submission of the script, **\$CASE.\$MACH.run**, the following will occur as part of a CCSM3 run:

- The **\$CASE.\$MACH.run** script is submitted and this script in turn executes the **\$CASE.\$MACH.build** script.
  - Restart files and restart pointer files located in `$DOUT_S_ROOT/restart/` are copied to the appropriate locations in the executable directory structure. If this is an initial run, these restart files will not exist, a warning message will be printed and **\$CASE.\$MACH.build** will continue.
  - A check is made to verify that `env_conf` has not been changed and that the tasks/threads settings in `env_mach.$MACH` have also not been modified since **configure** was invoked.
  - Necessary model input data is prestaged to each component executable directory.
- The CCSM model is run.
- Component model history, log, diagnostic, and restart files are copied to the short-term archive directory, `$DOUT_S_ROOT/` (by the **Tools/ccsm\_s\_archive** script).
- The `$DOUT_S_ROOT/restart/` directory is cleaned and populated with the latest restart data files and restart pointer files (by the **Tools/ccsm\_s\_archive** script).
- The restart files in `$DOUT_S_ROOT/restart/` are tarred up and placed in `$DOUT_S_ROOT/restart.tars/` with a unique filename. This is done by the **Tools/ccsm\_s\_archive** script.
- The long-term archiver, **\$CASE.\$MACH.I.archive**, is submitted to the batch queue if appropriate.
- **\$CASE.\$MACH.run** is resubmitted if the `$RESUBMIT` environment variable in `env_run` is greater than 0. The `$RESUBMIT` is then decremented by 1 as long as it is greater than 0.

In particular, the script, **\$CASE.\$MACH.run**, does the following:

- sources `env_conf`, `env_run` and `env_mach.$MACH`
- executes **\$CASE.\$MACH.build**
- sets up the local run environment
- runs the CCSM model
- copies log files back to `$LOGDIR/` if `$LOGDIR` is not set " "
- executes the short-term archive script, **Tools/ccsm\_s\_archive.csh** (if `$DOUT_S` is TRUE)
- submits the long-term archive script, **\$CASE.\$MACH.I.archive**, if:
  - **\$CASE.\$MACH.I.archive** exists for `$MACH`
  - `$DOUT_S` is TRUE
  - `$DOUT_LMS` is TRUE
- resubmits **\$CASE.\$MACH.run** if `$RESUBMIT` (in `env_run`) is greater than 0
- decrements `$RESUBMIT` by 1 if `$RESUBMIT` is greater than 0

## 5.4 Initialization types

The environment variable, `$RUN_TYPE` in `env_conf` determines the way in which a new CCSM run will be initialized. `$RUN_TYPE` can have values of `'startup'`, `'hybrid'` or `'branch'`.

In a startup run, each component's initialization occurs from some arbitrary baseline state. In a branch run, each component is initialized from restart files. In a hybrid run initialization occurs via a combination of existing CCSM restart files for some components (e.g. POP and CSIM) and initial files for other components (e.g. for CAM and CLM).

The value of `$START_DATE` in `env_conf` is ignored for a branch run, since each model component will obtain the `$START_DATE` from its own restart dataset. The coupler will then validate at run time that all the models are coordinated in time and therefore have the same `$START_DATE`. This is the same mechanism that is used for performing a restart run (where `$CONTINUE_RUN` set to TRUE). In a hybrid or startup run, `$START_DATE` is obtained from `env_conf` and not from component restart

or initial files. Therefore, inconsistent restart and/or initial files may be used for hybrid runs, whereas they may not be used for branch runs.

All CCSM components produce "restart" files containing data necessary to describe the exact state of the CCSM run when it was halted. Restart files allow the CCSM to be continued or branched to produce exactly the same answer (bit-for-bit) as if it had never stopped. A restart run is not associated with a new `$RUN_TYPE` setting (as was the case in CCSM2), but rather is determined by the setting of the environment variable `$CONTINUE_RUN` in `env_run`.

In addition to the periodic generation of restart files, some CCSM components (e.g. CAM and CLM) also periodically produce netCDF initial files. These files are smaller and more flexible than the component's binary restart files and are used in cases where it is not crucial for the new run to be bit-for-bit the same as the run which produced the initial files.

The following provides a summary of the different initialization options for running CCSM.

- startup - arbitrary initialization determined by components (default)
- hybrid - initialization occurs from the restart/initial files of a previous reference case, the start date can be changed with respect to reference case
- branch - initialization occurs from restart files of a previous reference case, cannot change start date with respect to reference case

Types of Files Used Under Various Runtype parameters:

	atm	lnd	ocn	ice	cpl
	-----	-----	-----	-----	-----
startup :	nc	internal	internal+file	binary	internal/delay
hybrid :	nc	nc	binary	binary	internal/delay
branch :	binary	binary	binary	binary	binary

Delay mode is when the ocean model starts running on the second day of the run, not the first. In delay mode, the coupler also starts without a restart file and uses whatever fields the other components give it for startup. It's generally climate continuous but produces initial changes that are much bigger than roundoff.

A detailed summary of each `$RUN_TYPE` setting is provided in the following sections.

#### 5.4.1 Startup runs

When the environment variable `$RUN_TYPE` is set to 'startup', a new CCSM run will be initialized using arbitrary baseline states for each component. These baseline states are set independently by each component and will include the use of restart files, initial files, external observed data files or internal initialization (i.e. a "cold start"). By default, the CCSM3.0 scripts will produce a startup run.

Under a startup run, the coupler will start-up using "delay" capabilities in which the ocean model starts running on the second day of the run, not the first. In this mode, the coupler also starts without a restart file and uses whatever fields the other components give it for startup.

The following environment variables in `env_conf` define a startup run:

- `$CASE` : new case name
- `$RUN_TYPE` : startup
- `$START_DATE`: YYYYMMDD (date for starting the run)

The following holds for a startup run:

- All models startup from some arbitrary initial conditions set in an external file or internally.
- The coupler sends the start date to the components at initialization.
- All models set the case name internally from namelist input.
- All models set the start date through namelist input or at initialization from the coupler.
- The coupler starts up in "delay" mode.

### 5.4.2 Hybrid runs

A hybrid run indicates that the CCSM is to be initialized using datasets from a previous CCSM run. A hybrid run allows the user to bring together combinations of initial/restart files from a previous CCSM run (specified `$RUN_REFCASE`) at a given model output date (specified by `$RUN_REFDATE`) and change the start date (`$RUN_STARTDATE`) of the hybrid run relative to that used for the reference run. In a branch run the start date for the run cannot be changed relative to that used for the reference case since the start date is obtained from each component's restart file. Therefore, inconsistent restart and/or initial files may be used for hybrid runs, whereas they may not be used for branch runs. For a hybrid run using the fully active component set (B) (see section 1.3.1), CAM and CLM will start from the netCDF initial files of a previous CCSM run, whereas POP and CSIM will start from binary restart files of that same CCSM run.

The model will not continue in a bit-for-bit fashion with respect to the reference case under this scenario. The resulting climate, however, should be continuous as long as no namelists or model source code are changed in the hybrid run. The variables `$RUN_REFCASE` and `$RUN_REFDATE` in `env_conf` are used to specify the previous (reference) case and starting date of the initial/restart files to be used. In a hybrid run, the coupler will start-up using the "delay" capabilities.

The following environment variables in `env_conf` define a hybrid run:

- `$CASE` : new case name
- `$RUN_TYPE` : hybrid
- `$START_DATE` : YYYYMMDD (date where to start this run)
- `$RUN_REFCASE`: reference case name (for initial/restart data)
- `$RUN_REFDATE`: YYYYMMDD (date in `$RUN_REFCASE` for initial/restart data)

Note that the combination of `$RUN_REFCASE` and `$RUN_REFDATE` specify the initial/restart reference case data needed to initialize the hybrid run. The following holds for a hybrid run:

- All models must be able to read a restart file and/or initial condition files from a different case and change both the case name and the start date internally to start a new case.
- The coupler must send the start date to the components at initialization.
- All models must set the case name internally from namelist input.
- All models must set the base date through namelist input or at initialization from the coupler.
- The coupler and ocean models start up in "delay" mode.

### 5.4.3 Branch runs

A branch run is initialized using binary restart files from a previous run for each model component. The case name is generally changed for a branch run, although it does not have to be.

In the case of a branch run, the setting of `$RUN_STARTDATE` in `env_conf` is ignored since each model component will obtain the start date from its own restart dataset. At run time, the coupler validates that all the models are coordinated in time and therefore have the same start date. This is the same mechanism that is used for performing a restart run (where `$CONTINUE_RUN` is set to TRUE).

Branch runs are typically used when sensitivity or parameter studies are required or when settings for history file output streams need to be modified. Under this scenario, the new case must be able to produce bit-for-bit exact restart in the same manner as a continuation run if no source code or namelist inputs are modified. All models must use full bit-for-bit restart files to carry out this type of run. Only the case name changes.

The following environment variables in `env_conf` define a branch run:

- `$CASE` : new case name
- `$RUN_TYPE` : branch

- `$RUN_REFCASE`: reference case (for restart data)
- `$RUN_REFDATE`: YYYYMMDD (date in `RUN_REFCASE` for restart data)

The following holds for a branch run:

- All models must be able to restart exactly from a branch run when compared to the equivalent continuation run given the same source code and namelist inputs.
- The base date set in the components must be read from a restart file.

#### 5.4.4 Prestaging input datasets for hybrid or branch runs

To start up a branch or hybrid run, restart and/or initial data from a previous run must be made available to each model component. As is discussed below, restart tar files of the form

```
$CASE.cesm.r.yyyy-mm-dd-sssss.id.tar
```

where `id` corresponds to a unique creation time stamp, are periodically generated. The restart tar files contain data that is required to start up either a hybrid or branch run.

The simplest way to make this data available to the hybrid or branch run at initialization is to untar appropriate reference case restart.tar file in the `$DOUT_S_ROOT/restart/` short-term archiving directory of the branch or hybrid run case. For example, assume that a new hybrid case, Test2, is to be run on machine blackforest, using restart and initial data from case Test1, at date `yyyy-mm-dd-sssss`. Also assume that the short-term archiving directory (`$DOUT_S_ROOT` (in `env_mach.blackforest`) is set to `/ptmp/$LOGNAME/archive/Test2`. Then the restart tar file

```
Test1.cesm.r.yyyy-mm-dd-sssss.id.tar
```

should be untarred in

```
/ptmp/$LOGNAME/archive/Test2/restart/
```

The script, `$CCSMROOT/scripts/ccsm_utils/Tools/ccsm_getrestart`, will then prestage this data to the Test2 component executable directories at run time.

## 5.5 Model output

CCSM3 is comprised of a collection of distinct models optimized for a very high-speed, parallel multi-processor computing environment. Each component produces its own output stream consisting of history, restart and output log files. Component history files are in netCDF format whereas component restart files are in binary format and are used to either exactly restart the model or to serve as initial conditions for other model cases.

Standard output generated from each CCSM component is saved in a "log file" located in each component's subdirectory under `$EXEROOOT/`. Each time the CCSM is run, a single coordinated timestamp is incorporated in the filenames of all output log files associated with that run. This common timestamp is generated by the run script and is of the form `YYMMDD-hhmmss`, where `YYMMDD` are the Year, Month, Day and `hhmmss` are the hour, minute and second that the run began (e.g. `ocn.log.040526-082714`). Log files can also be copied to a user specified directory using the variable `$LOGDIR` in `env_run`. The default is "", so no extra copy of the log file occurs.

By default, each component writes monthly averaged history files in netCDF format and also writes binary restart files. The history and log files are controlled independently by each component. Restart files, on the other hand, are written by each component at regular intervals dictated by the flux coupler via the setting of `$REST_OPTION` and `$REST_N` in `env_run`. Restart files are also known as "check-point" files. They allow the model to stop and then start again with bit-for-bit exact capability (i.e.

the model output is exactly the same as if it had never been stopped). The coupler coordinates the writing of restart files as well as the model execution time. All components receive information from the coupler and write restarts or stop as specified by the coupler. Coupler namelist input in `env_run` sets the run length and restart frequency via the settings of the environment variables `$STOP_OPTION`, `$STOP_N`, `$RESTART_OPTION` and `$RESTART_N`. Each component's log, diagnostic, history, and restart files can be saved to the local mass store system using the CCSM3 long-term archiver.

The raw history data does not lend itself well to easy time-series analysis. For example, CAM writes one large netCDF history file (with all the requested variables) at each requested output period. While this allows for very fast model execution, it makes it difficult to analyze time series of individual variables without having to access the entire data volume. Thus, the raw data from major CCSM integrations is usually postprocessed into more user-friendly configurations, such as single files containing long time-series of each output fields, and made available to the community (see section 10).

Archiving is a phase of the CCSM production process where model output is moved from each component's executable directory to a local disk area (short-term archiving) and subsequently to a long-term storage system (long-term archiving). It has no impact on the production run except to clean up disk space and help manage user quotas.

Short and long-term archiving environment variables are set in the `env_mach.$MACH` file. Although short-term and long-term archiving are implemented independently in the scripts, there is a dependence between the two since the short-term archiver must be turned on in order for the long-term archiver to be activated.

By default, short-term archiving is enabled and long-term archiving is disabled. Several important points need to be made about archiving:

- All output data is initially written to each component's executable directory (e.g. `$EXEROOT/atm/`, etc.).
- Unless a user explicitly turns off short-term archiving, files will be moved to the short-term archive area by default at the end of a model run.
- Users should generally turn off short term-archiving when developing new CCSM code.
- If long-term archiving is not enabled, users should monitor quotas and usage in the `$DOUT_S_ROOT/` directory and should manually clean up these areas on a regular basis.

### 5.5.1 Short-term archiving

Short-term archiving is executed as part of running the `$CASE.$MACH.run` script. The short-term archiving script, `ccsm_s_archive`, resides in the `ccsm_utils/Tools` (`$UTILROOT/Tools`) directory. Short-term archiving is executed after the CCSM run is completed if `$DOUT_S` is set to `TRUE` in `env_mach.$MACH`. The short-term archiving area is determined by the setting of `$DOUT_S_ROOT` in `env_mach.$MACH`.

The short-term archiver does the following:

- copies complete sets of generated restart/initial files and restart pointer files from each component's executable directory to `$DOUT_S_ROOT/restart/`.
- moves all history, log, diagnostic, restart and initial files from each component's executable directory to that component's specific directory under `$DOUT_S_ROOT/`
- tars up the contents of the directory `$DOUT_S_ROOT/restart/` and places the tarred set in the directory `$DOUT_S_ROOT/restart.tar/` with an appended unique date string

The `ccsm_s_archive` script is written quite generally. However, there may be certain user cases where it needs to be modified for a production run because different sets of files need to be stored. If this is the

case, **ccsm\_s\_archive** should be copied to the user's **\$CASEROOT/** directory and modified there since in general this file is shared among different production runs. In addition, the path to **ccsm\_s\_archive** in the **\$CASE.\$MACH.run** file also must be modified.

### 5.5.2 Long-term archiving

Long-term archiving is done via a separate CCSM script that can be run interactively or submitted in batch mode. Long-term archiving saves files onto the local mass store system. It also can copy data files to another machine via **scp**. Normally, the long-term archiver is submitted via batch automatically at the end of every CCSM production run. The long-term archive script is generated by **configure** and since is a machine-dependent batch script called **\$CASE.\$MACH.l\_archive**.

The environment variables which control the behavior of long-term archiving are set in the file, **env\_mach.\$MACH** (see section 4.8.5) and correspond to:

- **\$DOUT\_L\_MS**
- **\$DOUT\_L\_MSROOT**
- **\$DOUT\_L\_MSNAME** (optional, generally used at NCAR only)
- **\$DOUT\_L\_MSPWD** (optional, generally used at NCAR only)
- **\$DOUT\_L\_MSRPD** (optional, generally used at NCAR only)
- **\$DOUT\_L\_MSPRJ** (optional, generally used at NCAR only)

Not all of these parameters are used for all mass store systems. The long-term archiver calls **ccsm\_l\_archive** which in turns calls **ccsm\_mswrite** to actually execute the mass store writes. The script **ccsm\_mswrite** is configured to test the local mass store and execute the appropriate command to move data onto the local mass store. Both **ccsm\_l\_archive** and **ccsm\_mswrite** script reside in the **ccsm\_utils/Tools** (**\$UTILROOT/Tools/**) directory.

The long-term archiver is also capable of copying files to another machine or site via **scp**. This requires that **scp** passwords be set up transparently between the two machines and will also likely require modification to the **ccsm\_l\_archive** script to specify which files should be moved. The parameters in **env\_mach.\$MACH** that turn this on are:

- **\$DOUT\_L\_RCP**
- **\$DOUT\_L\_RCP\_ROOT**

The above feature is not currently supported.

Although the **ccsm\_l\_archive** script is written quite generally, there may be cases where it needs to be modified for a given production run because different sets of files need to be stored. If this is the case, **ccsm\_l\_archive** should be copied to the user's **\$CASEROOT/** directory, modified and the path to **ccsm\_l\_archive** in **\$CASE.\$MACH.run** also must be changed accordingly.

## 6 CCSM3 Use Cases

This section contains specific examples for creating and running various CCSM3 runs. We have selected examples that deal with issues that have most commonly effected CCSM3 users. Note that all examples in this section assume that short-term archiving is enabled (this will occur as a default setting in the `mach.env.$MACH` script). It is also assumed that the user will build the case directory in `/user/`.

### 6.1 Setting up a startup run

The following describes the steps necessary to set up a T42\_gx1v3 run with fully active model components. The case name for this run will be TestB. We will subsequently use this run as a reference case for setting up branch and hybrid runs in sections 6.2 and 6.3. TestB will start from 1989-01-01 and run for one year.

- `cd $CCSMROOT/ccsm3/scripts/`
- `invoke`  
**`create_newcase -case /user/TestB -mach $MACH -compset B`**  
 This will create the TestBcase directory in the `ccsm3/scripts/` directory.
- `edit env_conf:`
  - set `$CASESTR` to description of case
  - set `$RUN_TYPE` to startup
  - set `$RUN_STARTDATE` to 1989-01-01
  - set `$CSIM_MODE` to desired value (see section 4.6)
  - set `$IPCC_MODE` and `$RAMP_CO2_START_YMD` (if necessary) to desired value (see section 4.6)
- `edit env_run:`
  - set `$STOP_OPTION` to yearly
  - set `$RESUBMIT` to 0
- `run ./configure -mach $MACH`
- `run ./TestB.$MACH.build` interactively
- submit the **`TestB.$MACH.run`** run

CCSM3 contains the ability to automatically resubmit itself via the setting of the environment variable `$RESUBMIT` (see section 4.7). By setting `$RESUBMIT` to a value greater than 0, the **`TestB.$MACH.run`** will resubmit itself and decrement `$RESUBMIT` by 1. Setting `$RESUBMIT` to 0 (as is done above), will result in a year run that will not resubmit itself.

### 6.2 Setting up a branch run

The following describes the steps necessary to set up a branch (TestB.branch) run starting from reference case TestB at year 1990-01-01 (see section 6.1). The resulting branch run will run for a month and resubmit itself at the end of that run for another month.

In branch runs all model components start from binary restart files. Consequently, certain constraints apply to the reference cases that are used for starting branch runs. In general, the restart files used to start a branch run have to be consistent with the model version and model settings used in the branch run. In particular, restart files produced by older versions of the CCSM (e.g. CCSM2.0) will not be compatible with current versions of the model. Only restart files generated with the released CCSM3.0 code should be used in doing branch and hybrid runs.

Other restart issues to watch out for are the `$CSIM_MODE` setting (see below), the `$IPCC_MODE` setting (see below), and the number of CAM constituents. In setting up a branch run, if the atmospheric component is CAM, the number of atm CAM constituents (see the User's Guide to the NCAR Community Atmosphere Model 3.0 (CAM 3.0)) must be kept the same between the reference case and the

branch case. In particular, if *\$IPCC\_MODE* is set to OFF or RAMP\_CO2\_ONLY, then CAM will use 3 constituents. Otherwise, for all other values of *\$IPCC\_MODE*, CAM will use 11 constituents. Any run containing 3 CAM constituents may be branched from any other run containing 3 CAM constituents. Similarly, any run containing 11 CAM constituents may be branched from any other run containing 11 CAM constituents. As an example, a run with *\$IPCC\_MODE* set to 1870\_CONTROL may not be branched from a reference case where *\$IPCC\_MODE* is set to RAMP\_CO2\_ONLY.

Finally, when doing a branch run, the run start date is set by the each component's restart (i.e. *\$RUN\_STARTDATE* in *env\_conf* is ignored).

- cd *\$CCSMROOT/ccsm3/scripts/*
- run ***./create\_newcase -case /user/TestB\_branch -mach \$MACH -compset B***
- edit *env\_conf*:
  - set *\$CASESTR* to description of case
  - set *\$RUN\_TYPE* to branch
  - set *\$RUN\_REFCASE* to TestB
  - set *\$RUN\_REFDATE* to 1990-01-01
  - set *\$COMP\_ATM*, *\$COMP\_LND*, *\$COMP\_ICE* and *\$COMP\_OCN*
  - set *\$IPCC\_MODE*
  - set *\$CSIM\_MODE* (see below)
  - set *\$RAMP\_CO2\_START\_YMD* (see below)
- edit *env\_run*:
  - set *STOP\_OPTION* to nmonths
  - set *STOP\_N* to 1
  - set *RESUBMIT* to 1
  - (this will result in an 1 month initial run followed by a 1 month restart run)
- run ***./configure -mach \$MACH***
- place restart files in the short term archiving restart directory. For this case, untar the TestB restart tar file, *TestB.cesm.r.1990-01-01-00000.id.tar*, into the *\$DOUT\_S\_ROOT/restart/* directory. Note that *id* corresponds to a unique creation time stamp.
- run ***./TestB\_branch.\$MACH.build*** interactively
- submit the ***TestB\_branch.\$MACH.run*** run

*\$CSIM\_MODE* has two options; prognostic and oceanmixed\_ice. Restart files produced in oceanmixed\_ice mode cannot be used to restart CSIM in any other mode. On the other hand, oceanmixed\_ice mode can be initiated using other ice restart files if the CSIM *oml\_ice\_sst\_init* namelist parameter is set to true.

*\$RAMP\_CO2\_START\_YMD* sets the start of the CO2 ramping and should be set if *\$IPCC\_MODE* is set to RAMP\_CO2\_ONLY. If the branch run is from a RAMP\_CO2\_ONLY run, the start YMD should be set the same as in the baseline run. There are no tests in CCSM to verify continuity, and users can actually set the start *\$RAMP\_CO2\_START\_YMD* to anything they want.

### 6.3 Setting up a hybrid run

The following describes the steps necessary to set up a hybrid (TestB\_hybrid) run starting from initial/restart data from reference case TestB at year 1990-01-01 where the starting date is 1995-01-01. The hybrid branch run will run for a month and resubmit itself at the end of that run for another month.

Note that for hybrid runs, unlike branch runs, the run startdate in *env\_conf* is used.

- cd *\$CCSMROOT/ccsm3/scripts/*
- run ***./create\_newcase -case /user/TestB\_hybrid -mach \$MACH -compset B***
- edit *env\_conf*:

- set `$CASESTR` to description of case
- set `$RUN_TYPE` to hybrid
- set `$RUN_STARTDATE` to 1995-01-01
- set `$RUN_REFCASE` to TestB
- set `$RUN_REFDATE` to 1990-01-01
- set `$COMP_ATM`, `COMP_LND`, `COMP_ICE` and `$COMP_OCN`
- set `$IPCC_MODE`
- set `$CSIM_MODE` (see section 6.2)
- set `$RAMP_CO2_START_YMD` (see below)
- edit `env_run`:
  - set `$STOP_OPTION` to nmonths
  - set `$STOP_N` to 1
  - set `$RESUBMIT` to 1
  - (this will result in an 1 month initial run followed by a 1 month restart run)
- run `./configure -mach $MACH`
- place necessary restart files in the short term archiving restart directory by untarring the TestB restart tar file, `TestB.cesm.r.1990-01-01-00000.id.tar`, into the `$DOUT_S_ROOT/restart/` directory. Note that `id` corresponds to a unique creation time stamp.
- run `./TestB_hybrid.$MACH.build` interactively
- submit the `TestB_hybrid.$MACH.run` run

`$RAMP_CO2_START_YMD` sets the start date of the CO2 ramping and should be set if `$IPCC_MODE` is set to `$RAMP_CO2_ONLY`. If the reference case for the hybrid run is a `$RAMP_CO2_ONLY` run, the `$RAMP_CO2_START_YMD` should be set to the same value as in the reference case run. There are no tests in CCSM3.0 to verify continuity, and users can actually set the start `$RAMP_CO2_START_YMD` to anything they want.

## 6.4 Setting up a production run

To set up a production run, the user must run `create_newcase`, modify the `$CASEROOT/env_*` files as needed, run `configure`, interactively build the executables and submit the run script. We provide a recommended process below for doing this.

- `cd $CCSMROOT/ccsm3/scripts/`
- run `./create_newcase`
- `cd $CASEROOT/`
- edit `env_conf`:
  - set `$COMP_ATM`, `COMP_LND`, `COMP_ICE`, `COMP_OCN` and `$COMP_CPL`
  - set `$CSIM_MODE`
  - set `$GRID`
  - set `$RUN_TYPE` and `$RUN_STARTDATE`
  - set `$RUN_REFCASE` and `$RUN_REFDATE` if `$RUN_TYPE` is set to *branch* or *hybrid*
  - set `$IPCC_MODE` (and possibly `$RAMP_CO2_START_YMD`)
- edit `env_mach.$MACH`:
  - set `$DOUT_S` to `TRUE`
  - set `$DOUT_LMS` to `FALSE` (recommended)
  - set `$component NTASKS_XXX` and `$NTASKS_XXX` to non-default settings if needed
- edit `env_run`:
  - set `$STOP_OPTION` to nmonths (recommended)
  - set `$STOP_N` to the number of months you want to run
  - set `$RESUBMIT` to 0
  - (this will result in an nmonth initial run)

- run `./configure -mach $MACH`
- run `./$CASE.$MACH.build` interactively
- submit the `$CASE.$MACH.run` run
- after the run is complete,
  - edit `env_run`
    - \* set `$CONTINUE_RUN` to TRUE
    - \* set `$RESUBMIT` to a number greater than 1 (this will result in multiple automatic resubmissions)
  - edit `env_mach.$MACH`
    - \* set `$DOUT_LMS` to TRUE (this will start long term archiving)
    - \* set `$DOUT_LMSNAME` (currently only for NCAR MSS)
    - \* set `$DOUT_LMSPWD` (currently only for NCAR MSS)
    - \* set `$DOUT_LMSRPD` (currently only for NCAR MSS)
    - \* set `$DOUT_LMSPRJ` (currently only for NCAR MSS)
- submit the `$CASE.$MACH.run` run again

## 6.5 Monitoring a production run

Once a production run has started, there are several things that need to be monitored.

- Keep the `$RESUBMIT` flag in `env_run` to some integer value greater than 1. This flag is decremented each time the run is resubmitted and the run will automatically resubmit when `$RESUBMIT` is greater than 0. A number between 5 and 20 is typical.
- Verify that the long-term archiver is working properly by periodically reviewing the files in `$DOUT_S_ROOT/` and on the mass store.
- If `$LOGDIR` is not "" (see section 4.7), clean up the logs directory periodically.
- Monitor quotas in the scripts area and the short-term archive area. If a user goes over quota, the run will fail. Disk space usage will increase rapidly if the long-term archiver is not working properly.

## 6.6 Diagnosing a production run failure

A production run can fail for several reasons. A model component could "blow up", there could be a machine hardware or software failure, or something in the user's environment might cause the failure.

If a run fails, the following process can be used to clean up and diagnose the failure. When a run fails, it normally shuts down the run automatically. However, there are cases where it can continue to resubmit itself until the `RESUBMIT` flag is zero, then the run stops completely. This means that several runs may run after the failure. These will inevitably fail as well. The user will need to clean up extra runs, diagnose the problem, fix the problem, and restart the run.

First, identify the run that originally failed and remove all files created in subsequent run submissions if there are any. Files may need to be removed in `$LOGDIR/`, `$DOUT_S_ROOT/restart.tars` and the `stdout` and `stderr` files in `$CASEROOT/`. Next, look through the logs, `stdout`, and `stderr` files in the original run that failed. Search for the error strings in those files. See section 9.1 for more information about the log files. If an error can't be found, restart the run from the last good restart tar file (see section 6.7 for details). Set the `$RESUBMIT` value to zero to prevent run-away run. Run the single case and check whether the model failed at the same time, and diagnose the problem. Once diagnosed, restart the model and continue running as desired.

## 6.7 Restarting a production run

This case summarizes the steps to restart a production run. This might occur if a run fails, if restarts are corrupt, if a user needs to continue a run from a previous restart dataset, if files are scrubbed, or if a run is restarted after a period where it's been stopped. This step assumes the source code and scripts are available and addresses primarily the restaging of restart files.

- Identify the case to run and the restart date required. Verify scripts and source code exist.
- Delete all files in `$DOUT_S_ROOT/restart/`
- Untar the desired restart tar file into `$DOUT_S_ROOT/restart/` from the `restart.tars/` directory or another source. If the tar file is not available for the case or date desired, users will need to stage each required file individually from a mass store or disk. This also requires manual generation of each component's `rpointer` files.
- Set `$RESUBMIT` value in `env_run` to 0.
- Submit the run.

## 6.8 Handing off a production run to another person

The new CCSM3 scripts make it very simple to have more than one person run a production run. The following describes how to do this.

- `tar` or "`cp -r`" the original `$CASE/` directory to a new location. We will use `/home/user/$CASE/` for this new location in what follows.
- `cd /home/user/$CASE/`
- run **`./configure -cleanmach $MACH`**
- edit `env_run`
  - modify `$CCSMROOT`
  - modify `$CASEROOT`
  - make sure `$SETBLD` is TRUE
  - make sure `$RESUBMIT` is greater than 1
- run **`./configure -mach $MACH`**
- verify that the original and new scripts produce the same run scripts by doing a review (use `diff -r`, etc)
- create the new `$DOUT_S_ROOT/restart/` directory (see section `env_mach.$MACH`)
- populate the new `$DOUT_S_ROOT/restart/` directory with the current set of restart files generated by the old user
- interactively run **`./$CASE.$MACH.build`**
- submit **`./$CASE.$MACH.run`**

## 6.9 Generating IPCC configuration runs

To generate an IPCC configuration, modify the `$IPCC_MODE` setting in `env_conf` before running **`configure`**. The supported `$IPCC_MODE` options are

- OFF
- RAMP\_CO2\_ONLY
- 1870\_CONTROL

OFF is the default. The RAMP\_CO2\_ONLY option will be described below. It is not an IPCC configuration. All other options turn on IPCC features in the active components. These features include sulfates, volcanics, and greenhouse gas forcing. The 1870\_CONTROL is a constant 1870 spin-up. Changes to namelist, build, and scripts in general are resolved automatically when **configure** is run. The CAM **cam.buildnml\_prestage.csh** script is modified significantly when setting the *\$IPCC\_MODE* to an IPCC scenario.

To generate a “ramp\_co2” case, set the *\$IPCC\_MODE* value in *env.conf* to RAMP\_CO2\_ONLY and set the *\$RAMP\_CO2\_START\_YMD* value to an 8 digit yyymmdd start date. This is the date that the ramping will begin. When running **configure**, ramp\_co2 namelists will be included in the **Buildnml\_Prestage/cam.buldnml\_prestage.csh**.

There are additional CAM specific namelists that can be used to set the CO2 ramp slopes and caps. See the (see the User’s Guide to the NCAR Community Atmosphere Model 3.0 (CAM 3.0)) for more information. CAM namelist changes should be added manually to the **cam.buildnml\_prestage.csh** script as needed.

## 6.10 Adding a new machine to \$CCSMROOT/

Adding support into CCSM3 for a new machine is generally simple and straight forward. There is one file that must be modified, at most five new files that may need to be generated, and a seventh file that might be to be modified. In most all cases, existing files for a supported machine can be used as a starting point for the new machine.

The first file is a script used to verify that a machine name is valid. It is located at *\$CCSMROOT/scripts/ccsm\_utils/Tools/check\_machine*.

The next 5 files are new scripts to be located in the directory *\$CCSMROOT/scripts/ccsm\_utils/Machines/*. These set up a number of the path names, batch commands, and archive commands needed.

- *\$CCSMROOT/scripts/ccsm\_utils/Machines/batch.\** (REQUIRED) contains the machine specific commands needed to execute a batch or interactive job
- *\$CCSMROOT/scripts/ccsm\_utils/Machines/env.\** (REQUIRED) declares the number of processors for each component, a number of the default path names, and a number of default run time flags
- *\$CCSMROOT/scripts/ccsm\_utils/Machines/l\_archive.\** (OPTIONAL) contains the machine specific commands to support long term archiving
- *\$CCSMROOT/scripts/ccsm\_utils/Machines/modules.\** (OPTIONAL) contains the machine specific commands needed to declare modules
- *\$CCSMROOT/scripts/ccsm\_utils/Machines/run.\** (REQUIRED) contains the commands necessary to execute CCSM3 in the batch or interactive job

There is a naming convention associated with these files. The file suffixes contain the type of machine (ibm, sgi, linux, etc) and then the machine name. For example, the files for the machine bluesky are

- **batch.ibm.bluesky**
- **env.ibm.bluesky**
- **l\_archive.ibm.bluesky**
- **run.ibm.bluesky**

Note there is no *modules.ibm.bluesky* file. It is not needed for bluesky.

The seventh file that might need to be modified is in the *\$CCSMROOT/models/bld* directory. For each machine type there is a *Macros* files containing the architecture and machine specific build information including special compiler options, include pathnames, etc. For name of the file used for bluesky is *Macros.AIX*. Machine name specific information may be *ifdef*’d in this file. These files eliminate the need to modify any of the *Makefile*’s used by CCSM3.

The following process is recommended for creating scripts for a new machine. For demonstration, a concrete example is presented below where the new machine is an IBM power4 named mypower4.

- edit `$CCSMROOT/ccsm_utils/Tools/check_machine` and add the new machine name to the list named “resok”.  
For example, add mypower4 to the list.
- cd `$CCSMROOT/scripts/ccsm_utils/Machines/` and copy a set of machine specific files from a supported machine (whatever machine is “closest”) to files for the new machine. Use “ls \*\$MACH\*” to list machine specific files. For example,
  - cd `$CCSMROOT/scripts/ccsm_utils/Machines/`
  - ls \*bluesky\*
  - cp `env.ibm.bluesky env.ibm.mypower4`
  - cp `run.ibm.bluesky run.ibm.mypower4`
  - cp `batch.ibm.bluesky batch.ibm.mypower4`
  - cp `L_archive.ibm.bluesky L_archive.ibm.mypower4` (if needed)
- **START ITERATION**
  - edit the appropriate `Macros.*` file in `$CCSMROOT/models/bld` to modify default compilation settings
  - edit the three or four new \*mypower4 files as needed. At the very least,
    - \* edit the “set mach = “ parameter in `batch.ibm.mypower4` and `L_archive.ibm.mypower4`
    - \* edit `$EXEROOT`, `$DIN_LOC_ROOT`, `$DOUT_S_ROOT`, `$ARCH`, `$OS`, `$SITE`, and `$BATCHE` in `env.ibm.mypower4`
    - \* edit the queue parameters in the `L_archive.ibm.mypower4` and `batch.ibm.mypower4`.
    - \* edit the “run the model” section of `run.ibm.mypower4` if needed
  - cd `$CCSMROOT/scripts`
  - run `./create_newcase -case $CASEROOT -mach mypower4`
  - cd `$CASEROOT/`
  - run `./configure -mach mypower4`
  - review the generated scripts and try building and running the model on the new machine
  - if further modifications are required, type `./configure -cleanmach mypower4`
- return to **START ITERATION** above as needed.
- run appropriate pre-configure tests (see 7) to validate that CCSM3 behaves as expected on the new machine

The \*.L\_archive file is not absolutely required. Automated long-term archiving of data will simply not be accomplished without it. It’s likely that for an initial port, the L\_archive is not needed. However, it is recommended for production runs.

## 6.11 Modifying CCSM source code

There are a number of ways CCSM3 source code can be modified. Source code can be modified directly in the appropriate `$CCSMROOT/ccsm3/models/` component subdirectory. Alternatively, user-modified code can be placed in the appropriate `$CASEROOT/SourceMods/` component sub-directory.

The CCSM build process uses a search path to find files and then builds dependencies automatically. The `$CASEROOT/SourceMods/src.*` directories are always searched first for source files. These directories are provided as empty directories in `$CASEROOT/SourceMods/` by default. By using the `SourceMods/src.*` directories for user-modified code, the user has the ability to use a shared source code area (for the untarred CCSM3 source distribution) but have locally modified code.

To use the `$CASEROOT/SourceMods/src.*` directory, the user should copy over the relevant files into the `SourceMods/src.*` directory for the required component(s) before the build script is invoked, make sure `$SETBLD` is TRUE in `env_run`, then build and run the case.

## 6.12 Modifying input datasets

If the user wishes to use a subset of their own input datasets rather than those provided in `$DIN_LOC_ROOT/inputdata/`, they should do the following (as an example we assume that the user would like to replace the cam dataset

```
AerosolMass_V_64x128_clim_c031022.nc with mydataset_64x128_clim_c040909.nc
```

- `untar ccsm3.0.inputdata_user.tar` in a directory parallel to where the `$DIN_LOC_ROOT/inputdata/` directory is located
- place `mydataset_64x128_clim_c040909.nc` in `$DIN_LOC_ROOT_USER/inputdata_user/atm/cam2/rad/mydataset_64x128_clim_c040909.nc`
- `cd $CCSMROOT/ccsm3/scripts/`
- run `./create_newcase`
- `cd $CASEROOT/`
- run `./configure`
- edit `env_mach.$MACH` and set `$DIN_LOC_ROOT_USER` to point to the root directory containing `inputdata_user/`.
- edit `Builnml_Prestage/cam.buildnml_prestage.csh` and replace `set aeroptics = AerosolMass_V_64x128_clim_c031022.nc` with `set aeroptics = mydataset_64x128_clim_c040909.nc`
- interactively execute `./$CASE.$MACH.build`
- submit `./$CASE.$MACH.run`

## 7 Testing

This section describes some of the practical tests that can be used to validate model installation as well as verify that key model features still work after source code modifications. Several pre-configured test cases are included in the CCSM3.0 distribution. Their use is strongly encouraged. This section will be expanded as more pre-configured test cases are created.

**These tests are provided for validation and verification only and must not be used as starting points for production runs.** The user should always start a production run by invoking the `create_newcase` command.

### 7.1 Features to be tested

Below is a summary of some of the critical aspects of CCSM3 that are tested regularly as part of the CCSM development process. These tests are performed over a range of resolutions, component sets, and machines. For those features where pre-configured test cases exist, the first three letters of the test case names are specified.

- **Exact restart:** The ability to stop a run and then restart it with bit-for-bit continuity is key to making long production runs. This is tested by executing a 10 day startup run, writing restarts at the end of day 5, and then restarting the run from the component restart files produced at day 5 and running for an additional 5 days. The results at day 10 should be identical for both runs. Note that processor configuration (MPI processes, OpenMP threads, load balance) is not changed during these tests. Test case names begin with "ER".
- **Exact restart for branch runs:** The exact restart capability is tested in a similar way for branch runs. Test case names begin with "BR".
- **Exact restart for hybrid runs:** The exact restart capability is tested in a similar way for hybrid runs. Test case names begin with "HY".
- **Software Trapping:** Short CCSM runs are carried out regularly with hardware and software trapping turned on. Often, such trapping will detect code that produces floating-point exceptions, out-of-bounds array indexing and other run-time errors. Runs with trapping turned on are typically much slower than production runs, but they provide extra confidence in the robustness of the code. Trapping is turned on automatically in these test cases via setting `$DEBUG` to TRUE in `env_run`. Test case names begin with "DB".
- **Regression:** Often, a source code change is not expected to change results for one or more test case(s). It is always a good idea to test this expectation. A regression test runs a test case and compares results with a previous run of the same test case. The regression test fails if results are not identical. Regression testing can be enabled for any pre-configured test case when the test case is created.
- **Parallelization:** Several CCSM components produce bit-for-bit identical results on different processor configurations. Tests are carried out to verify this capability with many different processor counts using MPI, OpenMP, hybrid parallel, and no parallelization. Pre-configured test cases that compare runs made with different process configurations are under development.
- **Science Validation:** CCSM science validation takes several forms. First, the model should run for several months or years without any failures. Second, multi-year and multi-century runs are reviewed by scientists for problems, trends, and comparison with observations. Third, an automated climate validation test has been developed at NCAR that compares two multi-decadal runs for climate similarity. This test is performed as a part of port validation and for model changes. Pre-configured test cases for automated climate validation are under development.

## 7.2 Testing script

The **create\_test** script is used to create the pre-configured tests mentioned in section 7.1. A brief overview of **create\_test** follows.

### 7.2.1 create\_test

The script **create\_test** in `$CCSMROOT/scripts/` generates test cases automatically. **create\_test** was developed to facilitate rapid testing of CCSM3 under many different configurations with minimal manual intervention. Many of the scripts and tools used by **create\_newcase** are also used by **create\_test**. Following is a usage summary for **create\_test**:

```
method 1:
=====

create_test -test <testcase> -mach <machine> -res <resolution>
  -compset <component-set> [-testroot <test-root-directory>]
  [-ccsmroot <ccsm-root-directory>] [-pes_file <PES_file>]
  [-clean <clean_option>] [-inputdataroot <input-data-root-directory>]
  [-testid <id>] [-regress <regress_action> -regress_name <baseline_name>]
  [-baselineroot <baseline_root_directory>] ]

method 2:
=====

create_test -testname <full-test-name> [-testroot <test-root-directory>]
  [-ccsmroot <ccsm-root-directory>] [-pes_file <PES_file>] [-testid <id>]
  [-clean <clean_option>] [-inputdataroot <input-data-root-directory>]
  [-baselineroot <baseline_root_directory>]

help option:
=====

create_test -help
```

An up-to-date version of the above text can be generated by typing the **create\_test** command without any arguments. **create\_test** must be invoked from the `$CCSMROOT/ccsm3/scripts/` directory. Detailed descriptions of options and usage examples can be obtained via the **-help** option:

```
> create_test -help
```

Note that the **-ccsmroot** and **-testid** options are primarily used by developers to test **create\_test**. Normally these options should be avoided as incorrect use can lead to obscure errors.

In general, test cases are created for a supported machine, resolution and component-set combination. Test cases generated using the second method above use the standard CCSM3 shorthand test case naming convention. For example,

```
> create_test -testname TER.01a.T42_gx1v3.B.blackforest
  -testroot /ptmp/$USER/tst
```

creates an exact restart test, ER.01a, at T42\_gx1v3 resolution for component set B on machine blackforest. An equivalent command using the first method is:

```
> create_test -test ER.01a -mach blackforest -res T42_gx1v3 -compset B
  -testroot /ptmp/$USER/tst
```

Note that in using method 2, an extra “T” must prepend the testname in the “-test” argument. The **-testroot** option causes **create\_test** to create the test in the directory

```
/ptmp/$USER/tst/TER.01a.T42_gx1v3.B.blackforest.TESTID/
```

where TESTID is a unique integer string identifying the test. TESTID is generated automatically by **create\_test**. The intent of TESTID is to make it difficult to accidentally overwrite a previously created test with a new one.

If the **-testroot** option is not used, **create\_test** creates the test in the `$CCSMROOT/ccsm3/scripts/` directory. It is much safer to create tests in a completely separate directory. We recommend that the **-testroot** option always be used and that the test root directory always be outside of the `$CCSMROOT/` tree.

### 7.2.2 Using create\_test

The following steps should be followed to create a new test case. Sample commands follow most steps.

1. Go to the CCSM scripts directory `$CCSMROOT/ccsm3/scripts/`  

```
> cd $CCSMROOT/ccsm3/scripts/
```
2. Execute **create\_test**:  

```
> ./create_test -testname TER.01a.T42_gx1v3.B.blackforest
               -testroot /ptmp/$USER/tst
```

Note that **create\_test** prints the location of the test directory

```
...
Successfully created new case root directory \
    /ptmp/$USER/tst/TER.01a.T42_gx1v3.B.blackforest.163729
...
```

3. Go to the test directory:  

```
> cd /ptmp/$USER/tst/TER.01a.T42_gx1v3.B.blackforest.163729
```
4. Build the model interactively:  

```
> ./TER.01a.T42_gx1v3.B.blackforest.163729.build
```
5. Edit the test script to modify the default batch queue setting (optional):  

```
> vi ./TER.01a.T42_gx1v3.B.blackforest.163729.test
```
6. Submit the test script to the batch queueing system (using `llsubmit`, or `qsub`, or ...):

**Note that users must submit the test script**

```
TER.01a.T42_gx1v3.B.blackforest.163729.test
```

**NOT the run script**

```
TER.01a.T42_gx1v3.B.blackforest.163729.run.
```

```
> llsubmit ./TER.01a.T42_gx1v3.B.blackforest.163729.test
```

7. When the batch job completes, the result of the test will be written to file `Teststatus.out`. Examine `Teststatus.out` to find out if the test passed: If the last line of `Teststatus.out` is “PASS”, then the test passed. Otherwise, the test either failed or did not run.

A script, `batch.$MACH`, is created in `$CCSMROOT/ccsm3/scripts/` the first time **create\_test** is invoked for a test root directory. This file will contain both the interactive command to build the test and the batch submission command to run the test. Each time a new test is subsequently created in the same test root directory, build and run commands for the new test are appended to `batch.$MACH`. Thus, `batch.$MACH` can be used as follows to easily build and run a whole sequence of tests that share the same test root directory.

```
> cd /ptmp/$USER/tst
> ./batch.$MACH
```

The test scripts (TER.01a.T42\_gx1v3.B.blackforest.163729.test, etc.) should be modified prior to running `batch.$MACH` if the default queue is not adequate. Finally, if a new series of tests are to be created, `batch.$MACH` should be deleted before `create_test` is invoked again.

### 7.2.3 Available tests

The following list comprises the most commonly used tests. A full list of all available tests is obtained by running `create_test -help`.

- **SM.01a** = Smoke test.  
Model does a 5 day startup run.  
The test passes if the run finishes without errors.
- **ER.01a** = Exact restart test for startup run.  
Model does a 10 day startup test, writing a restart file at day 5. A restart run is then done starting from day 5 of the startup run of the initial 10 day run.  
The test passes if the restart run is identical to the last 5 days of the startup run.
- **ER.01b** = Same as ER.01a but with `IPCC_MODE` set to `1870_CONTROL`
- **ER.01e** = Same as ER.01a but with `IPCC_MODE` set to `RAMP_CO2_ONLY`
- **DB.01a** = Software trapping test.  
Model does a 5 day startup run where `$DEBUG` is set to `TRUE` in `env_run`.  
The test passes if the run finishes without errors.
- **BR.01a** = Branch run test.  
Model executes a startup reference run followed by branch run using the reference run restart output for initialization.  
The test passes if the branch run is identical to the reference run.
- **BR.02a** = Exact restart test of branch run.  
Model does a branch run test followed by an exact restart test of the branch run (executes a startup, branch, and continue run).  
The test passes if the restart run is identical to the last 5 days of the branch run.
- **HY.01a** = Hybrid run test.  
Model executes a startup reference run followed by a hybrid run using the reference run output for initialization.  
The test passes if the hybrid run finishes without errors (this is effectively a smoke test of a hybrid run).
- **HY.02a** = Exact restart test of hybrid run.  
Model executes a hybrid run followed by an exact restart test of the hybrid run (executes a startup, hybrid, and continue run).  
The test passes if the restart run is identical to the hybrid run.

Users are encouraged to provide feedback and suggestions for new CCSM tests by sending email to [csm@ucar.edu](mailto:csm@ucar.edu).

## 7.3 Common `create_test` use cases

The two most common applications of CCSM tests are described in more detail below.

### 7.3.1 Validation of an Installation

After installing CCSM3.0, it is recommended that users run the following suite of low-resolution tests before starting scientific runs. If these tests all pass it is very likely that installation has been done properly for the tested resolution and component set.

- **TER.01a.T31\_gx3v5.B**
- **TDB.01a.T31\_gx3v5.B**
- **TBR.01a.T31\_gx3v5.B**
- **TBR.02a.T31\_gx3v5.B**
- **THY.01a.T31\_gx3v5.B**
- **THY.02a.T31\_gx3v5.B**

The sample session below demonstrates how this test suite can be built and run on the IBM "bluesky" machine at NCAR. Note that when testing on machines located at other sites, the **-inputdataroot** option can be used to tell **create\_test** where to find input data sets. See the text generated by **create\_test -help** for more about this option. Also see section 7.3.2 for an example showing use of the **-inputdataroot** option.

1. Create a new directory for testing:
 

```
> mkdir -p /ptmp/$USER/tstinstall
```
2. Go to the CCSM scripts directory `$CCSMROOT/ccsm3/scripts/`

```
> cd $CCSMROOT/ccsm3/scripts/
```
3. Execute **create\_test** for each test case:
 

```
> ./create_test -testname TER.01a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
> ./create_test -testname TDB.01a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
> ./create_test -testname TBR.01a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
> ./create_test -testname TBR.02a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
> ./create_test -testname THY.01a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
> ./create_test -testname THY.02a.T31_gx3v5.B.bluesky
                  -testroot /ptmp/$USER/tstinstall
```
4. Go to the test directory:
 

```
> cd /ptmp/$USER/tstinstall/
```
5. Edit the test scripts to modify the default batch queue (optional):
 

```
> vi ./T*/*test
```
6. Build and submit all of the tests:
 

```
> ./batch.bluesky
```
7. Check test results.
 

When the batch jobs complete, look for test results in `T*/Teststatus.out`. If a test fails, then model installation may not be correct.

Users are advised to run additional test that specifically exercise features they plan to use (such as other component sets, model resolutions, dynamical cores, etc.).

### 7.3.2 Verification of Source Code Changes

When model source code is modified, several tests should be run to verify that essential model features have not been broken. The following suite should be run at a resolution of the users choosing:

- **TER.01a.\*.B**
- **TDB.01a.\*.B**
- **TBR.02a.\*.B**
- **THY.02a.\*.B**

Users are advised to run additional test that specifically exercise features they plan to use (such as other component sets, model resolutions, dynamical cores, etc.).

In some cases, source code modifications are not intended to cause any changes in model output. Examples include structural improvements or addition of new features that can be turned off at run time (or, less preferably, at build time). The regression testing features of **create\_test** can be used to quickly determine if model output has been changed unintentionally.

CCSM regression testing has two phases: baseline data set generation and comparison with a baseline data set. In the first phase, a test case is run using a "known-good" version of CCSM and model output is stored as a "baseline". A "known-good" version might be an unmodified copy of CCSM3.0, or it might be a version in which the user has confidence in due to previous testing. In the second phase, the same test case is run using newly modified CCSM source code and model output is compared with the "baseline". The test passes if there are no differences. Baseline data set generation and comparison are both enabled using the **-regress** and **-regress\_name** options to **create\_test**. Use **-regress generate** for baseline generation or **-regress compare** for baseline comparison. Every baseline data set must have a unique name. When generating a new baseline data set, use the **-regress\_name** option to specify the new name. When comparing with an existing baseline data set, use the **-regress\_name** option to specify an existing name.

The sample session below demonstrates baseline data set generation and comparison for a single test case, **ER.01a** using the IBM "bluesky" machine at NCAR. Note that when testing on machines located at other sites, the **-baselineroot** option can be used to tell **create\_test** where baseline data sets should be located. Also, the **-inputdataroot** option can be used to tell **create\_test** where to find input data sets. Both of these options are used in the sample session below. Also, see the text generated by **create\_test -help** for more about this option. In the sample session, assume that the "known-good" version is stored in `$CCSMROOTOK/ccsm3/` and the version under test is stored in `$CCSMROOT/ccsm3/`.

- Create a new directory for testing:
  - > `mkdir -p /ptmp/$USER/tstregress`
- **Create the baseline data set:**
  1. Create a new directory for the baseline data set:
    - > `mkdir -p /ptmp/$USER/mybaseline`
  2. Go to the CCSM scripts directory for the "known-good" version `$CCSMROOTOK/ccsm3/scripts/`:
  3. Execute **create\_test** with baseline generation: This use of the **-inputdataroot** option assumes that CCSM input data sets exist in `/ptmp/$USER/mybaseline/`.
    - > `create_test -test ER.01a -mach bluesky -res T31_gx3v5 -compset B`  
`-inputdataroot /ptmp/$USER/CCSM3.0/inputdata -regress generate`  
`-regress_name mynewbaseline -baselineroot /ptmp/$USER/mybaseline`  
`-testroot /ptmp/$USER/tstregress`
  4. Go to the test directory:
    - > `cd /ptmp/$USER/tstregress/TER.01a.T31_gx3v5.B.bluesky.G.130344/`
  5. Build the model interactively:
    - > `./TER.01a.T31_gx3v5.B.bluesky.G.130344.build`
  6. Edit the test script to select queue for batch submission (optional):

```
> vi ./TER.01a.T31_gx3v5.B.bluesky.G.130344.test
```

7. Submit the test script to the batch queue system (using `llsubmit`, or `qsub`, or ...):

```
> llsubmit ./TER.01a.T31_gx3v5.B.bluesky.G.130344.test
```

8. Check test results.

When the batch jobs complete, look for test results in `Teststatus.out`. If a test fails, then something is wrong with the "known-good" model (i.e. maybe it is not good after all). Fix the problem and start over from step 1. The new baseline data set will be stored in the specified baseline root directory with the specified name if and only if the ER.01a test case passes. Verify that model output was stored in the directory `/ptmp/$USER/mybaseline/mynewbaseline/TER.01a.T31_gx3v5.B.bluesky`.

- **Compare with the baseline data set:**

1. Go to the CCSM scripts directory for the version under test `$CCSMROOT/ccsm3/scripts/`:

2. Execute `create_test` with baseline comparison:

```
> create_test -test ER.01a -mach bluesky -res T31_gx3v5 -compset B
               -inputdataroot /ptmp/$USER/CCSM3.0/inputdata -regress compare
               -regress_name mynewbaseline -baselineroot /ptmp/$USER/mybaseline
               -testroot /ptmp/$USER/tstregress
```

3. Go to the test directory:

```
> cd /ptmp/$USER/tstregress/TER.01a.T31_gx3v5.B.bluesky.C.162532/
```

4. Build the model interactively:

```
> ./TER.01a.T31_gx3v5.B.bluesky.C.162532.build
```

5. Edit the test script to select queue for batch submission (optional):

```
> vi ./TER.01a.T31_gx3v5.B.bluesky.C.162532.test
```

6. Submit the test script to the batch queueing system (using `llsubmit`, or `qsub`, or ...):

```
> llsubmit ./TER.01a.T31_gx3v5.B.bluesky.C.162532.test
```

7. Check test results: When the batch jobs complete, look for test results in `Teststatus.out`. If baseline comparison failed, then the two models are not producing identical results.

Note that coupler log files are currently used for baseline comparison. Since these are ASCII text files it is possible, though unlikely, that a very small error appearing late in a test run might escape detection. The baseline data sets also include full-precision coupler history files. In a future patch these will be compared too to reduce chances of undetected bit-for-bit changes.

## 8 Performance

This section provides a brief overview of CCSM performance issues.

### 8.1 Compiler Optimization

As is discussed in section 5.2, compiler flags are set in the `$CCSMROOT/models/bld/Macros.*` files. The default flags were chosen carefully in order to produce exact restart and valid climate results for various combinations of processors, parallelization, and components. They were also chosen for performance and trapping characteristics as well as confidence. Many tradeoffs were made with respect to these choices. Users are free to modify the compiler flags in the Macros files. However, CCSM strongly suggests that users carry out a suite of exact restart tests, parallelization, performance, and science validations prior to using a modified Macros file for science.

Patches will be provided as machines and compilers evolve. There is no guarantee about either the scientific or software validity of the compiler flags on machines outside CCSM's control. The community should be aware that machines and compilers will evolve, change, and occasionally break. Users should validate any CCSM run on a platform before documenting the resulting science (see section 7).

### 8.2 Model parallelization types

Model parallelization types and accompanying capabilities critical for optimizing the CCSM load balance are summarized in the following table. For each component, a summary is provided of the component's parallelization options and if answers are bit-for-bit when running that component with different processor counts.

Table 6: CCSM3 Components Parallelization

Component Name	Version	Type	MPI Parallel	OpenMP Parallel	Hybrid Parallel	PEs bit-for-bit
cam	cam3	active	YES	YES	YES	NO
datm	datm6	data	NO	NO	NO	-
latm	latm6	data	NO	NO	NO	-
xatm	dead	dead	YES	NO	NO	YES
clm	clm3	active	YES	YES	YES	YES
dlnd	dlnd6	data	NO	NO	NO	-
xlnd	dead	dead	YES	NO	NO	YES
pop	ccsm_pop_1_4	active	YES	NO	NO	NO
docn	docn6	data	NO	NO	NO	-
xocn	dead	dead	YES	NO	NO	YES
csim	csim5	active	YES	NO	NO	NO
dice	dice6	data	NO	NO	NO	-
xice	dead	dead	YES	NO	NO	YES
cpl	cpl6	active	YES	NO	NO	YES

The following lists limitations on component parallelization:

- **cam** : can run on any number of processing elements (PES), decomposition is 1d in latitude, (generally use total pes that divides number of lats evenly or nearly evenly and less than or equal to the number of lats)
- **datm**: 1 PE only
- **latm**: 1 PE only

- **xatm**: 1d decomp, must divide grid evenly
- **clm**: can run on any number of pes
- **dlnd**: 1 PE only
- **xlnd**: 1d decomp, must divide grid evenly
- **pop** : 2d decomp, must divide the grid evenly
- **docn**: 1 PE only
- **xocn**: 1d decomp, must divide grid evenly
- **csim**: 2d decomp, must divide the grid evenly, does not run on 1 pe successfully
- **dice**: 1 PE only
- **xice**: 1d decomp, must divide grid evenly
- **cpl** : can run on any number of pes

Some architectures and compilers do not support OpenMP threads. Therefore, OpenMP threading may not be used on a given architectures even if a given component supports its use. Also, threading will not be supported on architectures that may support threading if your compiler does not support threading.

### 8.3 Load Balancing CCSM3

CCSM load balance refers to the allocation of processors to different components such that efficient resource utilization occurs for a given model case and the resulting throughput is in some sense optimized. Because of the constraints in how processors can be allocated efficiently to different components, this usually results in a handful of “sweet spots” for processor usage for any given component set, resolution, and machine.

CCSM components run independently and are tied together only through MPI communication with the coupler. For example, data sent by the atm component to the land component is sent first to the coupler component which then sends the appropriate data to each land component process. The coupler component communicates with the atm, land, and ice components once per hour and with the ocean only once a day. The overall coupler calling sequence currently looks like

Coupler

-----

```
do i=1,ndays    ! days to run
  do j=1,24     ! hours
    if (j.eq.1) call ocn_send()
    call lnd_send()
    call ice_send()
    call ice_recv()
    call lnd_recv()
    call atm_send()
    if (j.eq.24) call ocn_recv()
    call atm_recv()
  enddo
enddo
```

For scientific reasons, the coupler receives hourly data from the land and ice models before receiving hourly data from the atmosphere. Because of this execution sequence, it is important to allocate processors in a way that assures that atm processing is not held up waiting for land or ice data. It is easy to naively allocate processors to components in such a way that unnecessary time is spent blocking on communication and idle processors result.

While the coupler is largely responsible for inter-component communication, it also carries out some computations such as flux calculations and grid interpolations. These are not indicated in the above pseudo-code.

Since all MPI “sends” and “receives” are blocked, the components might wait during the send and/or receive communication phase. Between the communication phases, each component carries out its internal computations. In general, a components time loop looks like:

#### General Physical Component

```
-----
do i=1,ndays
  do j=1,24
    call compute_stuff_1()
    call cpl_rcv()
    call compute_stuff_2()
    call cpl_snd()
    call compute_stuff_3()
  enddo
enddo
```

So, compute\_stuff\_1 and compute\_stuff\_3 are carried out between the send and the receive, and compute\_stuff\_2 is carried out between the receive and send. This results in a communication pattern that is represented below. We note that for each ocean communication, there are 24 ice, land, and atm communications. However, aggregated over a day, the communication pattern can be represented schematically below and serves as a template for load balancing CCSM3.

```
ocn  r-----s
     ^
ice  ^   r   s   |
     ^   ^   |   |
lnd  ^ r  ^   |   s   |
     ^ ^ ^   |   |   |
atm  ^ ^ ^   |   |   r   s |
     ^ ^ ^   v   v   ^   v v
cpl  s-s--s---r---r---s-----r-r
      time->
```

```
s = send
r = rcv
```

CCSM3 runtime statistics can be found in coupler, csim, and pop log files whereas cam and clm create files of the form `timing.<processID>` containing timing statistics. As an example, near the end of the coupler log file, the line

```
(shr_timer_print) timer 2: 1 calls, 355.220s, id: t00 - main integration
```

indicates that 355 seconds were spent in the main time integration integration loop. This time is also referred to as the “stepon” time. Simply put, load balancing involves reassigning processors to components so as to minimize this statistic for a given number of processors. Due to the CCSM processing sequences, it is impossible to keep all processors 100% busy. Generally, a well balanced configuration will show that the atm and ocean processors are well utilized whereas the ice and land processors may

indicate considerable idle time. It is more important to keep the atm and ocean processors busy as the number of processors assigned to atm and ocean is much larger than the number assigned to ice and land.

The script `getTiming.csh`, in the directory `$CCSMROOT/scripts/ccsm_utils/Tools/getTiming`, can be used to aid in the collection of run time statistics needed to examine the load balance efficiency.

The following examples illustrate some issues involved in load balancing a CCSM3 run for a T42\_gx1v3 run on bluesky.

Case	LB1	LB2
OCN cpus	40	48
ATM cpus	32	40
ICE cpus	16	20
LND cpus	8	12
CPL cpus	8	8
total CPUs	104	128
stepon	336	280
node seconds	34944	35840
simulated yrs/day	7.05	8.45
simulated yrs/day/cpu	.067	.066

In the above example, adding more processors in the correct balance resulted in an ensemble that was "faster" (computed more years per wall clock day) and statistically just as efficient (years per day per cpu). The example below shows that assigning more processors to a given run may speed up that run (generates more simulated years per day) but may be less processor efficient.

Case	LB3	LB4
OCN cpus	32	48
ATM cpus	16	40
ICE cpus	8	20
LND cpus	4	12
CPL cpus	4	8
total CPUs	64	128
stepon	471	280
node seconds	30144	35840
simulated yrs/day	5.03	8.45
simulated yrs/day/cpu	.078	.066

Learning how to analyze run time statistics and properly assign processors to components takes considerable time and is beyond the scope of this document.

## 9 Troubleshooting (Log Files, Aborts and Errors)

This section describes some common aborts and errors, how to find them, and how to fix them. General suggestions are provided below. Model failures occur for many reasons, however, not the least of which is that the science is “going bad”. Users are encouraged to constantly monitor the results of their runs to preempt failures. A few of the most frequent errors and aborts are documented below.

### 9.1 Error messages and log files

CCSM3 does not always handling log and error messages in an optimal manner since numerous challenges exist in gracefully handling messages for a multiple executable model. We hope to fully address this problem in the future. For now, there are several items to be aware of:

- There are many files where messages from components or scripts might be generated.
  - stdout files associated with the batch job, usually found in the `$CASEROOT/` directory
  - stderr files associated with the batch job, usually found in the `$CASEROOT/` directory
  - log files associated with each component, usually found in the `$EXEROOT/$COMPONENT/` directory
  - `fort.*` files generated by a component, usually found in the `$EXEROOT/$COMPONENT/` directory
- Error messages can be found in any of the above files
- The messages might NOT be written out properly ordered by either time or processor (due to model parallelization).
- Messages from one processor sometimes overwrite or truncate messages from other processors when writing to the same output file
- There is no consistent error writing convention in CCSM across components
- Errors associated with the scripts are normally written to either the stdout or stderr file
- Error keywords sometimes found in the log files are “error, abort, warn, or endrun”. “`grep -i`” for these keywords in the above sources to start to isolate an error message
- Aborts due to system problems can result in no error messages. System errors are often created by nfs mounts dropping, processors or nodes failing in the middle of a run, or network or other hardware failures.
- Error messages don’t always appear only at the end of an output files. Users should start by searching the end of output files, but then should progress to reviewing the entire log files for critical messages.

### 9.2 General Issues

There are a number of general issues users should be constantly aware of that can cause problems.

- disk quotas: jobs will fail if a disk quota is exceeded due to either bytes or inodes.
- queue limitations: jobs will fail if they exceed time or resource limits associated with a machine and batch environment
- disk scrubbing: jobs will fail if critical files are scrubbed. Many environments employ scrubbers in addition to quotas to control disk usage. Users should be aware of scrubbing policy and monitor their disk usage.

### 9.3 Model time coordination

An error message like “ERROR: models uncoordinated in time” is generated by the coupler and indicates that the individual component start dates do not agree. This occurs when an inconsistent set of restart files are being. Users should confirm that the restart files used are consistent, complete, and not corrupted. If users are trying to execute a branch run, they should review the constraints associated with a branch run and possibly consider a hybrid run if inconsistent restart files are desired.

### 9.4 POP ocean model non-convergence

An error message like XX indicates the POP model has failed to converge. See the POP model users guide for more information. This typically occurs when the CFL number is violated in association with some unusually strong forcing in the model.

The solution to this problem is often to reduce the ocean model timestep. This pop timestep is set in the **Buildnml\_prestage/pop.buildnml\_prestage.csh** file. The value associated with DT\_COUNT should be increased in the **sed** command. DT\_COUNT indicates the number of timesteps per day taken in POP. A higher number of DT\_COUNT means a shorter timestep. Initially, users should increase DT\_COUNT by 10 to 20 percent then restart their run. This will slow the POP model down by an equivalent amount. The results will not be bit-for-bit but should be climate continuous. Users can reset the timestep as needed to trade-off model stability and performance.

### 9.5 CSIM model failures

An error message like XX is from the ice model and usually indicates a serious model problem. These errors normally arise from bad forcing or coupling data in CSIM. Users should review their setup and model results and search for scientific problems in CSIM or other models.

### 9.6 CAM Courant limit warning messages

An error message like

```
COURLIM: *** Courant limit exceeded at k,lat= 1 62 (estimate = 1.016),
```

in the CAM log file is typical. This means CAM is approaching a CFL limit and is invoking an additional internal truncation to minimize the impact. These warning messages can be ignored unless a model failure has occurred.

### 9.7 CAM model stops due to non-convergence

The CAM model has been observed to halt due to high upper level winds exceeding the courant limit associated with the default resolution and timestep. The atmosphere log file will contain messages similar to:

```
NSTEP = 7203774 8.938130967099877E-05 7.172242117666898E-06 251.2209.84438
COURLIM: *** Courant limit exceeded at k,lat= 1 62 (estimate = 1.016)
solution has been truncated to wavenumber 41 ***
*** Original Courant limit exceeded at k,lat= 1 62 (estimate = 1.016) ***
```

If changes have been introduced to the standard model, this abort may be due to the use of too large of a timestep for the changes. The historical solution to this problem is to increase the horizontal diffusivity or the `kmxhdc` parameter in CAM for a few months in the run then reset it back to the original value. However, this solution has impact on the climate results. If changes such as these are undesirable, please contact the CAM scientific liaison at NCAR for further assistance.

To change the horizontal diffusivity, add the CAM namelist variable HDIF4 to the CAM namelist in **Buildnml\_prestage/cam.buildnml\_prestage.csh** and set it to a value that is larger than the default value for the current case. The default value can be found in the CAM log file.

Finally, if the failure occurs in the first couple of days of an startup run, the CAM namelist option DIVDAMPN can be used to temporarily increase the divergence dampening (a typical value to try would be 2.).

## 9.8 T31 instability onset

At T31 resolution (e.g. T31\_gx3v5), strong surface forcing can lead to instability onsets that result in model failure. This has been know to occur in several paleo climate runs. This failure has been linked to the interaction of the default model timestep in CAM and CLM (1800 secs) and the default coupling interval between CAM and CLM (1 hour). A solution to this problem has been to decrease the CAM and CLM model timesteps to 1200 seconds. The user also needs to reset the RTM averaging interval, `rtm_nsteps`, in the CLM namelist from 6 to 9 to be consistent with the current namelist defaults.

## 10 Post Processing

A completed run of CCSM3.0 will produce a large number of files, each with many variables. Post processing these files presents several unique challenges. Various techniques and tools have been developed to meet these challenges. Most involve software that does accompany the CCSM3 distribution. The user will have to download and install these tools separately. In the following sections, some of these tools will be described and their download location identified. Following sections will demonstrate various techniques using these tools.

### 10.1 Software Overview

#### 10.1.1 The netCDF Operators (NCO)

The netCDF Operators are command line executables specifically designed for multi-file bulk processing. They are free, run on numerous operating systems, and are essential in reducing the number of CCSM files to a manageable level. While not officially supported, they are widely used and easy to learn. The operators and on line user's manual can be found at:

[nco.sourceforge.net](http://nco.sourceforge.net)

Table 7 lists the operators and gives a brief description. The user is directed to the web page listed above for specific details, limitations, and usage.

Table 7: netCDF Operators

Operator Name	Function
<b>ncap</b>	arithmetic processor
<b>ncatted</b>	attribute editor
<b>ncbo</b>	binary operator (add, subtract, multiply, divide)
<b>ncea</b>	ensemble averager
<b>nccat</b>	ensemble concatenator
<b>ncflint</b>	file interpolator
<b>ncks</b>	kitchen sick (variable extraction)
<b>ncra</b>	record averager
<b>ncrcat</b>	record concatenator
<b>ncrename</b>	renamer
<b>ncwa</b>	weighted averager

Specific examples of these operators exist in sections [10.2.1](#) and [10.2.2](#).

#### 10.1.2 The NCAR Command Language (NCL)

NCL is a free, interpreted computer language developed at NCAR. It is highly suited to meteorological and climatological data processing and visualization. It runs on UNIX, AIX, IRIX, LINUX, MACOS X, and Windows under cygwin. It handles netCDF, GRIB, and HDF data formats with ease and is officially supported. The following web site contains a large on line library of graphical and data processing scripts that can be downloaded and applied:

[www.cgd.ucar.edu/csm/support/](http://www.cgd.ucar.edu/csm/support/)

Questions about the software should be directed to [ncl-talk@ucar.edu](mailto:ncl-talk@ucar.edu) or to Sylvia Murphy ([murphys@ucar.edu](mailto:murphys@ucar.edu)). NCL exists on all NCAR supercomputers and CGD systems. It can be also be downloaded at:

[ngwww.ucar.edu/ncl/download.html](http://ngwww.ucar.edu/ncl/download.html)

Questions about the NCL binaries and their installing should be directed to Mary Haley ([haley@ucar.edu](mailto:haley@ucar.edu)) of the Scientific Computing Division.

### 10.1.3 Component Model Processing Suite (CMPS)

CMPS is similar in functionality to the netCDF operators (see section 10.1). They are command line executables that are available on NCAR's supercomputers. Unlike the NCO, CMPS only works with CCSM model data. An on line user's guide is available at:

[www.cgd.ucar.edu/csm/support/FH/CMPS/index.shtml](http://www.cgd.ucar.edu/csm/support/FH/CMPS/index.shtml).

Table 8 lists the current operators and gives a brief description of their function. CMPS is officially supported. The user is advised to visit the web page above or contact Sylvia Murphy ([murphys@ucar.edu](mailto:murphys@ucar.edu)) for more information and specific usage.

Table 8: Component Model Processing Suite

Operator Name	Function
add.scp	addition
diff.scp	difference/anomalies
e2d.scp	extract all 2D variables
esvc.scp	extract selected variables and concatenate
have.scp	horizontal averaging
iso.scp	iso-level extraction
iso_interp.scp	interpolation of a variable to an iso surface
mult.scp	multiplication
rmse.scp	RMSE
subext.scp	sub-region extraction
ttext.scp	time series extraction
ttest.scp	t-test
topbot.scp	single level extraction
trans.scp	transects
vert.scp	vertical averaging
volave.scp	volume averaging
vert.scp	vertical averaging
zave.scp	zonal averaging

### 10.1.4 Diagnostic Packages

Comprehensive diagnostic packages exist for each model component. Not all are publicly available. CCSM3 diagnostic packages are designed to do bulk file processing (e.g. concatenate files and average using the NCO), conduct the necessary data processing, and output NCL graphics in a web format. Table 9 contains a list of the currently available packages, their point of contact, and download location.

Table 9: Publicly available diagnostics

Component	POC	Download location
atm	Mark Stevens ( <a href="mailto:stevens@ucar.edu">stevens@ucar.edu</a> )	<a href="http://www.cgd.ucar.edu/cms/diagnostics/">www.cgd.ucar.edu/cms/diagnostics/</a>

### 10.1.5 Commercial Tools

People who run the CCSM use many commercial tools for post processing. These can include IDL, Fortran, MatLab, Yorick, and Vis5D. These are mentioned for information only. Use of these tools is not supported, and it is up to the user to download, install, and debug their own programs.

## 10.2 Post Processing Strategies

There is no way this guide can provide a detailed set of instructions for all of the post-processing tasks a typical CCSM3.0 run requires. What follows is a short discussion of some of the more common tasks.

### 10.2.1 Selected Variable File Concatenation

CCSM output consists of many files each containing many variables. Often researchers are only interested in a subset of those variables. One strategy for file management is to extract the variables of interest and concatenate them into a smaller subset of files. The netCDF operators (see section 10.1) are well suited to this task. The following is a code snippet from a c-shell script that uses **ncks** to extract the selected variables T,U, and V and then uses **ncrcat** to combine the files together.

```
*****
# extract variables using ncks
*****
ls *.nc > file.txt                # get the list of files
set fnlst = 'cat file.txt'
foreach x ($fnlst)                # loop over each file
  ncks -A -h -v T,U,V $x temp_$x  # extract variables to temp file
end
*****
# concatenate variables with ncrcat
*****
ncrcat -A -h temp_*.nc concatenated.nc # concatenate all temp files together
*****
```

### 10.2.2 Creating Climatologies

Climatologies can be created using the netCDF operators (see section 10.1) or NCL (see section 10.1.2). The code snippet below demonstrates the use of **ncra** to create annual averages for a set of years using monthly data for those years. **ncra** is then used again to average the annual averages to create an annual climatology.

```
*****
#      CALC YEARLY ANNUAL AVERAGES FROM MONTHLY DATA
*****
@ yr_cnt      = $yr_strt                # create year counter
@ yr_end      = $yr_strt + $num_yrs - 1 # put space btwn "-" and "1"
while ( $yr_cnt <= $yr_end )           # loop over years
  set yr_prnt = 'printf "%04d" {$yr_cnt}' # format(change as required)

  ls ${DIR}${case}_${yr_prnt}*.nc > file.txt # get list of files
  set files = 'cat file.txt'
  ncra -O $files ${DIR}${case}_${yr_prnt}_ann.nc # create annual average
end while
```

```

*****
#      CREATE CLIMATOLOGICAL ANNUAL AVERAGES
*****
ls ${DIR}${case}*ann.nc > file.txt           # get list of ann average
set files = `cat file.txt`
ncra -O $files ${DIR}${case}_ANN_climo.nc   # create climatological avg
*****

```

There is a web page devoted to creating climatologies using NCL

[www.ncl.ucar.edu/Applications/climo.shtml](http://www.ncl.ucar.edu/Applications/climo.shtml)

This page demonstrates just a few of the climatological related functions available. For a full list, see

[www.ncl.ucar.edu/Document/Functions/climo.shtml](http://www.ncl.ucar.edu/Document/Functions/climo.shtml)

### 10.2.3 Atmospheric Hybrid to Pressure Level Interpolation

The atmospheric model output is on hybrid coordinates (see the User's Guide to the NCAR Community Atmosphere Model 3.0 (CAM 3.0) for details). Often it is desirable to convert these to pressure coordinates. The following web page demonstrates this conversion using NCL (see section 10.1.2):

[www.ncl.ucar.edu/Applications/vert.shtml](http://www.ncl.ucar.edu/Applications/vert.shtml)

### 10.2.4 POP remapping

A special NCL (see section 10.1) library script (**pop\_remap.ncl**) has been developed for POP remapping. This library script comes with the NCL distribution. In order to remap POP, special POP weight and grid files are necessary. **SCRIP**, which was developed at Los Alamos National Labs, is used to create these files. NCAR maintains a repository of grid and weight files for model configurations that have been scientifically validated (see section 1.5). If a map or weight file for one of these validated configurations is desired, contact Sylvia Murphy ([murphys@ucar.edu](mailto:murphys@ucar.edu)) to have the files placed on an anonymous ftp server. The following web site describes how to use **pop\_remap.ncl** and NCL to create other POP graphics:

[www.gfdl.ucar.edu/csm/support/CSM\\_Graphics/pop.shtml](http://www.gfdl.ucar.edu/csm/support/CSM_Graphics/pop.shtml)

## 11 Glossary

- **archive** - a phase of the CCSM production process in which model output is moved from the executable directory to a local disk before being saved to the local long-term storage system. This encompasses both short and long term archiving. See also **ccsm\_archive** or **L.archive**.
- **ARCH** - machine architecture, set in `env_mach`.
- **BATCH** - machine batch system, set in `env_mach`.
- **branch** - a specific `RUN_TYPE` that starts the model from a complete restart set from a previous run.
- **build script** - The C-shell scripts which builds the CCSM. In the CCSM3 distribution, this script is created by `configure` for a specific machine.
- **CASE** - experiment or case name.
- **ccsm\_archive** - a script that carries out short term archiving of model output. See also **archive**.
- **CASEROOT** - full path to the scripts directory (often this is `$CCSMROOT/scripts/$CASE`).
- **CCSMROOT** - full path to the top directory of the source code release.
- **COMP\_** - Defines a specific CCSM component. There are five CCSM components, ATM, LND, ICE, OCN, CPL which are set in `env_conf`. This parameter is resolved in CCSM script generation.
- **component** - Each CCSM model (except for the coupler) is represented by at least three CCSM components. The CCSM components are `cam`, `latm`, `datm`, `xatm`, `clm`, `dlnd`, `xlnd`, `csim`, `dice`, `xice`, `pop`, `docn`, `xocn`, `cpl`.
- **component set** - A unique set of CCSM3 component combinations. Often designated by a single letter like A, B, F, G, M, or X.
- **CONTINUE\_RUN** - Sets the run type. A run is either a continue run or it will be a `RUN_TYPE` run. See **RUN\_TYPE**
- **Delay Mode** - Runs of CCSM where the ocean model starts on the second day of the run. This is the default way of starting up for startup and hybrid runs.
- **DIN\_LOC\_ROOT** - full path to the inputdata directory.
- **DOUT\_S** - logical that turns on short term archiving.
- **DOUT\_S\_ROOT** - full path to the short term archival directory. See also **archive** and **ccsm\_archive**.
- **DOUT\_L** - logical that turns on long term archiving.
- **DOUT\_S\_ROOT** - full path to the long term archival directory on the mass store.
- **EXEROOT** - full path to the top directory where CCSM is going to run.
- **GRID** - overall CCSM grid combination. Set in `env_conf`. GRID is a resolved parameter in script generation.
- **hybrid** - a specific `RUN_TYPE` that starts the model from restart and initial datafiles from previous runs (hybrid mode). Can also refer to combined MPI/OpenMP parallelization paradigm (hybrid parallel).

- **L\_archive** - See **L\_archive script**.
- **L\_archive script** - the long term archiver script. a phase of the CCSM production process in which model output is moved from local disk to the local long-term storage system. The **L\_archive** is `$CASE.$MACH.L_archive`.
- **MACH** - machine name, typically blackforest, seaborg, moon, etc.
- **model** - There are five in CCSM3: atmosphere, land, sea-ice, ocean, and coupler. See also **component**. See also **component** or **component set**.
- **NTASK\_** - the number of MPI tasks for each model, set in the `env_mach` script.
- **NTHRD\_** - the number of OpenMP threads for each MPI task for each model. Set in the `env_mach` script.
- **OS** - local operating system, typically AIX, IRIX64, etc.
- **resolution** - The grid, see also **GRID**
- **resolved scripts** - Scripts that are generated for a specific configuration. Configuration settings that are resolved include resolution (grid), components, `RUN_TYPE`, and machine. If any of these items is changed, new scripts must be generated.
- **RESUBMIT** - an integer flag in `env_run`. If it's greater than zero, the run will be resubmitted after completion and the flag will be decremented.
- **RUN\_TYPE** - Sets the initialization type for the given case. This variable should not be confused with `CONTINUE_RUN` which is a flag which determines whether the given case is restarted or not. This is set in `env_conf` and is a resolved variable in the generation of scripts. `RUN_TYPE` can have values of `STARTUP`, `BRANCH`, or `HYBRID`. The initial run is executed when `CONTINUE_RUN` is `FALSE`. If `CONTINUE_RUN` is `TRUE`, then the run operates in continue mode. See **CONTINUE\_RUN**.
- **run script** - The C-shell scripts which runs the CCSM. In the CCSM3 distribution, this script is created by `configure` for a specific machine.
- **sandbox** - A users working directory. This term is typically used when referring to a users local copy of a version of CCSM under development. A shared source code area is NOT a sandbox.
- **script or scripts** - The term script refers to two different things in CCSM. First, there are scripts that are used to generate files that will run CCSM. These are things like `create_newcase`, `configure`, and files that exist in the `ccsm_utils` directory. Second, scripts also refer to the files that build, run, and archive CCSM. These are also called run scripts. Those files are usually named `$CASE.$MACH.[build/run/L_archive]` In the most sense, CCSM scripts generate CCSM run scripts.
- **script or scripts directory** - The directory named `scripts` in the `ccsm3` directory. Scripts required to create a CCSM3 case are here. Typically, CCSM3 run scripts and cases are also located in this directory.
- **SITE** - location, typically NCAR, LANL, ORNL, NERSC, etc. The `env_mach` script sets this automatically.
- **startup** - a specific `RUN_TYPE` that starts the model from some arbitrary initial conditions.