

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

# CCSM2.0.1 User's Guide

*Lawrence Buja and Tony Craig*

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

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Upgrades from CCSM2.0 release	4
1.1.1	Linux platform	4
1.1.2	Grids and Model Resolutions	4
1.1.3	Data Models	4
1.1.4	Model Bugs Detected and Corrected	5
1.2	The CCSM component models	5
1.3	The component models: CAM, POP, CSIM, CLM	6
1.4	Supported resolutions, configurations and platforms	6
<b>2</b>	<b>CCSM2.0.1 Quick Start Guide</b>	<b>8</b>
2.1	What is needed to run CCSM2?	8
2.2	Downloading and untarring the CCSM2.0.1 distribution	8
2.3	The CCSM source code	9
2.4	Input datasets	10
2.5	Building the CCSM	10
2.6	Running the CCSM	11
2.7	CCSM output data	11
2.7.1	Output log files	11
2.7.2	Binary history and restart output data	12
<b>3</b>	<b>The CCSM Scripts</b>	<b>13</b>
3.1	The CCSM Run Script: test.a1.run	13
3.1.1	Set batch system options	13
3.1.2	Define the common run environment	15
3.1.3	Select multi-processing and resolution specs	22
3.1.4	Run the setup script for each component	22
3.1.5	Run the CCSM integration	26
3.1.6	Archive and harvest	28
3.1.7	Resubmit	29
3.2	Sample Component Setup Script: cpl.setup.csh	30
3.2.1	Document the setup script	30
3.2.2	Set local component script variables	31
3.2.3	Define and position the input datasets	32
3.2.4	Write the input namelist	34
3.2.5	Build the component executable	37
<b>4</b>	<b>Building the CCSM</b>	<b>38</b>
4.1	The CCSM directory structure	38
4.2	\$CSMBLD contents	38
4.3	\$EXEDIR structure	39
4.4	Input data positioning	40
4.5	Building the model interactively	40
<b>5</b>	<b>Running the CCSM</b>	<b>41</b>
5.1	Start up	41
5.1.1	Running a simple test case of the fully coupled model on the NCAR IBM machine, blackforest	41
5.1.2	Changing the configuration	41
5.1.3	Changing the RUNTYPE	43

5.1.4	Running on a new machine . . . . .	43
5.1.5	Getting into production . . . . .	44
5.2	What is the <code>ccsm_joe</code> file? . . . . .	45
5.3	How does auto-RESUBMIT work? . . . . .	45
5.3.1	Runaway jobs . . . . .	45
5.4	Batch queuing challenges . . . . .	45
5.5	Modifying source code . . . . .	46
5.6	CCSM Data Management . . . . .	46
5.7	What is harvesting doing? . . . . .	46
5.8	Monitoring the integration . . . . .	47
5.9	Data processing . . . . .	47
5.10	Comparing output to NCAR controls . . . . .	47
<b>6</b>	<b>Testing the CCSM</b> . . . . .	<b>47</b>
6.1	Exact restart test . . . . .	47
<b>7</b>	<b>Common Aborts, Errors, Debugging, and Performance Issues</b> . . . . .	<b>48</b>
7.1	Common aborts and errors . . . . .	48
7.1.1	Component model has trouble building . . . . .	48
7.1.2	Model won't continue due to restart problem . . . . .	48
7.1.3	Ocean model stops due to ocean non-convergence or time-stepping problem . . . . .	48
7.1.4	Ice model stops due to ice mpdata transport instability . . . . .	49
7.1.5	Atmosphere model stops due to ocean non-convergence or time-stepping problem . . . . .	49
7.2	Debugging . . . . .	49
7.3	Performance issues . . . . .	50
<b>8</b>	<b>Supporting Scripts:</b> . . . . .	<b>51</b>
8.1	Experiment dependent scripts . . . . .	51
8.2	Scripts in the tools directory . . . . .	51
<b>9</b>	<b>The Graphical User Interface: <code>ccsm_gui</code></b> . . . . .	<b>53</b>
<b>10</b>	<b>The Atmosphere Setup Script: <code>atm.setup.csh</code></b> . . . . .	<b>55</b>
10.1	Document the atmosphere setup script . . . . .	55
10.2	Define and position the atmosphere input datasets . . . . .	56
10.3	Define resolution dependent parameters . . . . .	58
10.4	Write the atmosphere input namelist . . . . .	59
10.5	Create the atmosphere executable . . . . .	61
<b>11</b>	<b>The Ocean Model Setup Script: <code>ocn.setup.csh</code></b> . . . . .	<b>65</b>
11.1	Document the ocean setup script . . . . .	65
11.2	Set code location and resolution dependencies . . . . .	66
11.3	Position initial files . . . . .	67
11.4	Build the executable . . . . .	68
11.5	Parse the date variables . . . . .	71
11.6	Modify <code>pop_in</code> . . . . .	71
11.7	Define the ocean input datasets . . . . .	72
11.8	Position the ocean input datasets . . . . .	74

<b>12 The Sea-Ice Model Setup Script: ice.setup.csh</b>	<b>76</b>
12.1 Document the sea-ice setup script . . . . .	76
12.2 Set the sea-ice model configuration flags . . . . .	77
12.3 Acquire the sea-ice initial and boundary files . . . . .	77
12.4 Create the sea-ice namelist input file . . . . .	80
12.5 Build the sea-ice model executable . . . . .	83
<b>13 The Land Model Setup Script: lnd.setup.csh</b>	<b>87</b>
13.1 Document the land setup script . . . . .	87
13.2 Set the land model configuration flags . . . . .	88
13.3 Create the namelist input file . . . . .	91
13.4 Create the land model executable . . . . .	93
<b>14 The Data Model Setup Scripts: d***.setup.csh</b>	<b>96</b>
14.1 Create the namelist input file . . . . .	98
<b>15 Glossary</b>	<b>101</b>

## 1 Introduction

The Community Climate System Model (CCSM) is a coupled climate model for simulating the earth's climate system. Composed of four separate model components 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.

Both high- and low-resolution versions of the CCSM have been developed. The high-resolution version is best suited for simulating near-past, present-day and future climate scenarios, while the low-resolution option is commonly used for paleoclimate research and debugging runs. The October 2002 CCSM2.0.1 release specifically supports the both high-resolution (T42 atmosphere/land and gx1v3 ocean/sea-ice) and low-resolution (T31 atmosphere/land and gx3v4 ocean/sea-ice) model configurations for IBM S/390 and SGI Origin 2000 platforms.

The CCSM project is a cooperative effort by the 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 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).

### 1.1 Upgrades from CCSM2.0 release

The CCSM2.0.1 release gives exactly the same results (bit-for-bit) as the CCSM2.0 release. However, several upgrades have been incorporated into CCSM2.0.1

#### 1.1.1 Linux platform

CCSM2.0.1 has been run on a Linux platform and the CCSM run script now includes options for building a Linux version of the CCSM. The Linux version of the CCSM has not yet been extensively exercised, so additional user modifications may be required to obtain optimal performance.

#### 1.1.2 Grids and Model Resolutions

New Coarse-Resolution "Paleo" Resolution Supported in CCSM2.0.1

The CCSM2.0 release officially supported only one resolution, referred to as "T42\_gx1v3," which was used in the CCSM2.0 Control Run. The CCSM2.0.1 release supports a second, coarser resolution, "T31\_gx3v4," which has been validated in a paleo control run. A sample setup for the paleo run can be found in the CCSM2.0.1 release, in the `scripts/test.a2` directory.

Obsolete Grid Eliminated from CCSM2.0.1

All references to the older, unsupported coarse resolution, T31\_gx3, which was never scientifically validated, have been eliminated from CCSM2.0.1

Description of Supported Grid Resolutions

Please refer to the CCSM2.0.1 FAQ home page for a discussion on the specifics of the various model grids and resolutions.

#### 1.1.3 Data Models

Data-Model Input Datasets Updated

The input datasets for the data models were, in many cases, either outdated, incomplete, or both. The following datasets have been updated in the CCSM2.0.1 data models:

**datm** – The forcing datasets have been updated. Two 10-year "F" (active atmosphere, active land, thermodynamic ice, and data ocean model) cases were run for 10 years, one at the "T42\_gx1v3" resolution and one at the new "T31\_gx3v4" resolution. The last five years of atmosphere-model output are now available for use by the datm model. The datm.setup.csh script has been modified to cycle through only one of these years, but all five are available from the NCAR Mass Store.

**dlnd** – The forcing datasets have been updated. The last five years of the 10-year "F" runs described above are now available for use by the dlnd model. The dlnd.setup.csh script has been modified to cycle through only one of these years, but all five are available from the NCAR Mass Store.

**dice** – The ice-fraction dataset has been updated to use the datasets based upon those created by Jim Hurrell. The ice-fraction data are combined with corresponding sea-surface temperature data into a single file. The data are on a regular, 1-degree latitude/longitude grid. The dice.setup.csh script has been modified to reference this dataset.

**docn** – The sea-surface temperatures from the combined sea-surface temperature/ice-fraction dataset described above is used by the data ocean model. The docn.setup.csh script has been modified to reference this dataset.

#### 1.1.4 Model Bugs Detected and Corrected

**Active Atmosphere Model (CAM)** – CAM has an array-allocation problem that does not affect any CCSM2.0 coupled run configurations. It has been fixed in the stand-alone model and in CCSM2.0.1. (reported May 17, 2002)

CAM has an array-allocation problem in ccsm\_msg.F90. It has been fixed in CCSM2.0.1. (reported May 22, 2002)

**Active Ocean Model (POP)** – There is a bug in hmix\_gm.F in CCSM2.0 POP, which results in not insignificant, but small changes in the results. In tests, the average boundary layer depth changes by 2 - 2.5m in the convection regions in the GIN and Weddell Seas. The average heat flux into the ocean after 30 years is reduced by 0.09 W/m\*\*2. Changes in the barotropic streamfunction are  $\pm 1$ Sv everywhere, and there are very small changes in the meridional overturning and poleward heat flux.

A patch which corrects this error is available on the CCSM2.0.1 download site (patch1). The correction to hmix\_gm.F is implemented by default in the test.a2 T31/gx3v4 case, but the patch has not been applied generally because it changes the answers of the control case. Users who are not trying to re-create the control case should apply the patch before running.

There is a bounds overflow in a physics routine in the CCSM2.0, but only in an option that is not turned on by default. Thus, the overflow did not affect the CCSM2.0 control run. The error occurs when kappa\_choice = 'variable' is chosen in the namelist input. The error has been fixed in CCSM2.0.1.

##### Correction in Reading Restart Files

The CCSM2.0 version will not read b20.007 control run restart files. This has been fixed in CCSM2.0.1.

## 1.2 The CCSM component models

The CCSM consists of four dynamical geophysical models linked by a central coupler. The components are:

<b>cpl</b>	The Coupler
<b>atm</b>	The Atmospheric component
<b>ocn</b>	The Ocean component
<b>ice</b>	The Sea-ice component
<b>lnd</b>	The Land-surface component

During the course of a CCSM integration, each of the four component models integrate forward in time simultaneously, periodically stopping to exchange information with the coupler. The coupler

receives fields from the component models, computes, maps and merges this information and sends the fields back to the other component models. By brokering this sequence of communication interchanges, the coupler manages the overall time progression of the coupled model.

### 1.3 The component models: CAM, POP, CSIM, CLM

Both full dynamical model and data-cycling versions are supplied for each model component. The dynamical models are the complete, fully interactive climate system component models, such as the POP ocean model or the CAM atmosphere general circulation model. The data-cycling models are small, simple codes which simply read static datasets and supply that data to the coupler. The fast and inexpensive data-cycling components can be easily used in place of any of the expensive dynamical models for testing purposes. The CCSM are written in FORTRAN 90 to optimize performance.

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 longitude by 64 latitude 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. More information on the CAM can be found at [www.cesm.ucar.edu/models/ccsm2.0.1](http://www.cesm.ucar.edu/models/ccsm2.0.1)

The ocean model is an extension of the Parallel Ocean Program (POP) from Los Alamos National Laboratory (LANL). POP grids in CCSM are displaced pole (Greenland Pole) grids at approximately 1-degree (gx1v3) and 3.6-degree (gx3v4) horizontal resolutions. The POP web page can be found at [www.cesm.ucar.edu/models/ccsm2.0.1/pop](http://www.cesm.ucar.edu/models/ccsm2.0.1/pop)

The sea-ice component of CCSM is the Community Sea-Ice Model (CSIM4). 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. The CSIM web page is [www.cesm.ucar.edu/models/ccsm2.0.1/csim](http://www.cesm.ucar.edu/models/ccsm2.0.1/csim)

The Community Land Model is the land model for both the CCSM and the un-coupled version of CAM. It is 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 CLM web page is [www.cgd.ucar.edu/models/clm2](http://www.cgd.ucar.edu/models/clm2)

The CCSM components are joined by the Coupler. The Coupler controls the rate of model execution and the interchange of all data between the different components. The coupler home page is at [www.cesm.ucar.edu/models/ccsm2.0.1/cpl5](http://www.cesm.ucar.edu/models/ccsm2.0.1/cpl5)

The dynamical models can consume substantial amounts of memory and CPU time while producing large volumes output data. The data-cycling versions of the components simply read existing datasets that were previously written by the dynamical models and pass these data to the coupler. These data-cycling components are very inexpensive to run and produce no output data. For these reasons, the data components are used for both test runs and certain types of model simulation runs. Information on the data models can be found under the CCSM2.0.1 release page: [www.cesm.ucar.edu/models/ccsm2.0.1](http://www.cesm.ucar.edu/models/ccsm2.0.1)

### 1.4 Supported resolutions, configurations and platforms

The current release has been validated on the IBM power3 and SGI O2K class of machines. In addition, the current release runs successfully on Compaq hardware, although it currently does not restart exactly. This will be fixed shortly.

A long control simulation has been carried out with the fully active CCSM configuration at T42 atmosphere and land resolution and gx1v3 ocean and ice resolution. This configuration is scientifically validated and model output will be released.

In addition, a number of additional configurations and resolutions are available with this release. The release version contains three different atmosphere components (atm, datm, latm), two different

land models (lnd, dlnd), two different ocean models (ocn, docn), two different sea-ice models (ice, dice), and a single coupler (cpl). These components can be mixed and matched to carry out various climate experiments. NCAR does not guarantee the scientific validity of any configuration except the fully active configuration for which a long control was carried out. For the atmosphere and land components, both T42 and T31 resolutions are available. For the ocean and ice components, both gx1v3 and gx3v4 resolutions are available. For the latm component, a T62 resolution is supported. In general, there is a shorthand naming convention for configuration setups. These are

- A = datm,dlnd,docn,dice,cpl
- B = atm,lnd,ocn,ice,cpl
- C = datm,dlnd,ocn,dice,cpl
- D = datm,dlnd,docn,ice,cpl
- F = atm,lnd,docn,ice (prescribed ice mode),cpl
- G = latm,dlnd,ocn,ice,cpl
- H = atm,dlnd,docn,dice,cpl
- I = datm,lnd,docn,dice,cpl
- K = atm,lnd,docn,dice,cpl
- M = latm,dlnd,docn,ice (mixed layer ocean mode), cpl

In summary, this release contains the following configurations:

Platforms

- fully supported: IBM, SGI
- partly supported: Linux, CPQ

Configurations

- atmosphere: atm (cam2), datm (datm5), latm (latm5)
- land: lnd (clm2), dlnd (dlnd5)
- ocean: ocn (pop), docn (docn5)
- ice: ice (csim4), dice (dice5)
- cpl: cpl (cpl5)

Resolutions

- atmosphere, land: T42, T31
- ocean, ice: gx1v3, gx3v4
- latm: T62

and the following table summarizes the tested configurations (only the T42\_gx1v3 / B case has been scientifically validated):

	A	B	C	D	F	G	H	I	K	M
T42_gx1v3	*	*	*	*	*		*	*	*	
T31_gx3v4	*	*	*	*	*		*	*	*	
T62_gx1v3						*				*
T62_gx3v4						*				*

\* indicates tested configuration



## 2 CCSM2.0.1 Quick Start Guide

This section briefly describes how to download the CCSM2.0.1 source code and input datasets, how to configure the model, how to build and run the model, and what output is created. The process can be summarized as follows:

- Download the source code and input datasets from the CCSM2.0.1 website:  
www.cesm.ucar.edu/models/ccsm2.0.1
- Untar the source code into your home directory (or similar)
- Untar the input datasets into a large disk area
- Create a new case by using the CCSM graphical user interface (GUI) found in the \$HOME/ccsm2.0.1/scripts/gui\_run directory or by manually copying and reconfiguring the example CCSM scripts in the \$HOME/ccsm2.0.1/scripts/test.a1 directory to a new directory
- Modify the copy of the CCSM scripts for your local environment
- Submit the job
- Review the model logs files and output

The amount of effort to get CCSM running depends largely on the similarity of your site compared to the NCAR environment. This quick start guide is most applicable to users that are running the model at NCAR. Modifying the scripts to run at other may require a number of small changes to the CCSM scripts and build procedure.

### 2.1 What is needed to run CCSM2?

Two target architectures are supported: IBM and SGI. Linux version is also offered, but control runs have not yet been carried out on this platform.

- OS: IBM AIX, SGI IRIX64 or Linux
- Compilers: Fortran 90, C
- Tools: gunzip, gnumake, Perl5 or greater, Perl/Tk for gui\_run
- Permanent disk space and Temporary disk space
- Libraries: MPI, netCDF
- Input data : 1.6 GB for T31\_gx3v4 grid, 0.8 GB for T42\_gx1v3 grid
- CCSM2 source code: 35 MB

### 2.2 Downloading and untarring the CCSM2.0.1 distribution

CCSM2.0.1 is available via the web from: [www.cgd.ucar.edu/csm/models/ccsm2.0.1](http://www.cgd.ucar.edu/csm/models/ccsm2.0.1)

The CCSM2.0.1 distribution consists of the following files:

- **ccsm2.0.1.tar.gz** CCSM2.0.1 source codes, run scripts and documentation
- **ccsm2.0.1.inputdata.T31\_gx3v4.tar** CCSM2.0.1 input data for T31\_gx3v4 resolution
- **ccsm2.0.1.inputdata.T42\_gx1v3.tar** CCSM2.0.1 input data for T42\_gx1v3 resolution
- **ccsm2.0.1.inputdata.T62.tar** CCSM2.0.1 latm data for T62 resolution
- **ccsm2.0.1.inputdata.CLM\_RAW\_SFC\_GEN\_FILES.tar** are optional files needed to generate a CLM surface dataset

By default, the CCSM2.0.1 distribution is configured to run on the NCAR IBM SP blackforest under the directory /ptmp/\$LOGNAME/\$CASE To uncompress and untar the file ccsm2.0.1.tar.gz, use the Unix gunzip and tar commands:

```
gunzip -c ccsm2.0.1.tar.gz | tar -xf -
```

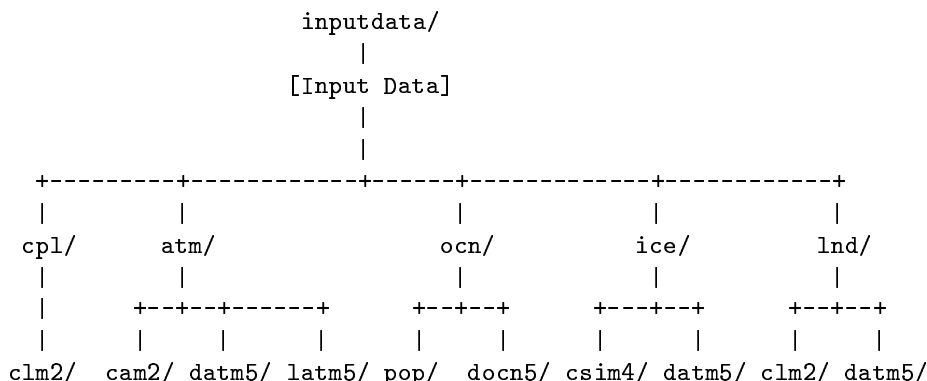
To untar the data files:

```
tar -xf ccsm2.0.1.inputdata.T42_gx1v3.tar
```



## 2.4 Input datasets

The directory tree for the tarfile containing the CCSM2.0.1 input data looks like



The initial and boundary datasets for each component will be found in their respective subdirectories. The input data should be untarred in a relatively large disk area. It could take up several Gigabytes of disk space depending which input datasets are downloaded.

## 2.5 Building the CCSM

Two levels of C-shell scripts are used to build and run the model. A single run script (i.e. test.a1.run) coordinates the building and running the complete system while the component setup scripts (i.e. atm.setup.csh or ocn.setup.csh) are responsible for building each CCSM component model.

A CCSM run is controlled by a master C-shell script, referred to as the “run script”. By convention, the name of the script is the case name (\$CASE) with a suffix of “.run”. For example, if the CASE name was “test.a1”, the run script would be named “test.a1.run”. In the CCSM2.0.1 distribution, the main run script is scripts/test.a1/test.a1.run

The run script has three main tasks:

- Define environment variables for building and running the coupled system
- Run the setup script for each component (see next section)
- Execute all components simultaneously

The common build and run environment variables are set in the run script and are automatically propagated to each of the component model setup scripts. These variables define such things as the machine architecture, the number of CPUs to run on and common file-naming conventions.

Once the master run script defines the common environment, each of the component models are built using the component setup scripts. For each component, the setup script will:

- Position any initial or boundary datasets needed by that component
- Generate the component’s namelist input file
- Build the component executable

Finally, once all the component setup scripts have successfully been run, the run script executes all components simultaneously.

Two model resolutions are supported. At high resolution, the atmosphere and land model grids are approximately 2.8 degrees latitude by longitude (T42 spectral truncation) with 26 levels in the vertical for the atmosphere, while the ocean and ice grids are approximately 1 by 1 degrees in the ocean. At low resolution, the atmosphere/land and ocean/ice grids are roughly 3.75 by 3.75 degrees (T31) at 26 levels in the vertical and 3 by 3 degrees respectively.

## 2.6 Running the CCSM

To get the CCSM up and running, a new CASE should be created and some script variables need to be modified.

- Create the case name and copy the contents of the test.a1 (or test.a2 for the paleoclimate version) into the new case directory. The examples here will show test.a1.

```
cd /home/\$LOGNAME/ccsm2.0.1/scripts
cp -r test.a1/* newcase/
```

- Modify the main run script

```
cd /home/\$LOGNAME/ccsm2.0.1/scripts/newcase
mv test.a1.run newcase.run
mv test.a1.har newcase.har
edit newcase.run
  change "job\_name" to newcase
  change "setenv CASE" to "newcase"
  change "setenv CASESTR" to a useful string
  change "setenv CSMROOT" to /home/\$LOGNAME/ccsm2.0.1
  change "setenv CSMDATA" to the local path to the inputdata directory
  change "setenv EXEROOT" to the local directory where the model will run
  change "setenv ARCRROOT" to the local directory for archiving model output
```

- Run the script

submit /home/\\$LOGNAME/ccsm2.0.1/scripts/newcase/newcase.run to the batch queue.

Operationally, a new CCSM case is started as either a **startup**, **hybrid** or **branch** run, depending on the science requirements. Wall-clock limits in the batch queues restrict all runs to a finite length, usually 1-3 model years. At specified intervals during the run (usually annually), restart and initial files will be created by all component models in a coordinated fashion.

After the first **startup**, **hybrid** or **branch** run successfully completes, the run is extended forward in time by resubmitting the run script to the batch queues as a **continue** run. This is done simply by changing the RUNTYPE setting to "continue" in the \$SCRIPTS/case.run file and resubmitting the job. The continuation run reads in the restart files created by the previous run and steps forward in time. The continuation run is then resubmitted as many times as necessary to extend the case to the desired length. Scientific integrity requires that the continuation runs must produce exactly the same answer as if the model had been run continuously without stopping.

## 2.7 CCSM output data

The CCSM must be viewed as a collection of distinct models optimized for very high-speed, parallel multi-processor computing. This results in raw output data streams from each component which do not present the raw data in the most user-friendly manner. Raw data from major CCSM integrations is usually postprocessed into user-friendly configurations, such as single files containing long time-series of each output field.

### 2.7.1 Output log files

The printed output from each CCSM component is saved in a "log file" in the respective component subdirectories under \$EXEROOT. Each time the CCSM is run, a single coordinated timestamp is incorporated in the filenames of all the 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 (i.e. \$EXEROOT/ocn/ocn.log.000626-082714).

### **2.7.2 Binary history and restart output data**

The binary output data are written from each CCSM component independently.

By default, CCSM2.0.1 writes monthly averaged history files for all components in netCDF format. CCSM2.0.1 also writes out binary restart files from all components at regular intervals. The total output volume of the model output can vary greatly depending upon the output frequencies and options selected.

The raw history data can be analyzed, but traditionally, the raw data package does not lend itself well to easy time-series analysis. For example, the atmosphere dumps all the requested variables into one large file at each requested output period. While this allows for very fast model execution, this makes it impossible 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 user-friendly configurations, such as single files containing long time-series of each output fields and made available to the community.

### 3 The CCSM Scripts

Three levels of C-shell scripts are used to build and run the model. The run script coordinates the building/running the complete system, the component setup scripts are responsible for configuring each individual CCSM component model, and the tool scripts handle generic operations, such as file transfer and positioning.

The CCSM execution is controlled by a single script, referred to as the “run script”. In the standard CCSM distribution, this file is `$HOME/ccsm2.0.1/scripts/test.a1/test.a1.run`. An alternative script, called `test.a2`, is supplied to demonstrate using the paleoclimate version of CCSM. The `test.a1` script will be described here. By convention, the name of the script is the name of the CASE with a suffix of “.run”. For example, if the CASE name was “test.01”, the run script would be named “test.a1.run”, located in the scripts directory, `$SCRIPTS`.

Once the run script has defined the common environment, each of the component models (`cpl`, `atm`, `ocn`, `ice` `ld`) is configured using a component setup scripts. The common build and run environment defined in the run script is automatically propagated to each of the component model setup scripts. These variables define such things as the machine architecture, the number of CPUs to run on, and common experiment and file naming conventions.

Finally, when all of the setup scripts have successfully completed, the run script executes all CCSM components simultaneously.

#### 3.1 The CCSM Run Script: `test.a1.run`

The coordinated build and execution of the CCSM is controlled by a single UNIX C-shell script. The example script distributed with CCSM is called `test.a1.run`. This script has the following tasks:

- a. Set batch system options
- b. Define build and run environment variables common to all components.
- c. Select multi-processing and resolution specifications
- d. Run the setup script for each component (see next section)
- e. Run the CCSM Integration.
- f. Archive/harvest/resubmit when this run is finished

Below, the various steps of the `test.a1.run` scripts are outlined.

##### 3.1.1 Set batch system options

```
#!/bin/csh -f
#=====
# CVS $Id: CCSM.Scripts.tex,v 1.13 2002/10/10 19:16:58 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/CCSM.Scripts.tex,v $
# CVS $Name: $
```

The first section of the CCSM run scripts documents the revision control version of the script. The Concurrent Versions System (CVS) is used as the revision control software for the CCSM. The “CVS Name:” should be used when reporting CCSM problems.

```

#=====
# This is a CCSM coupled model NQS batch job script
#=====
#-----
# a. Set batch system options
#-----

# ----- NCAR IBM SP: blackforest -----
# @ shell = /usr/bin/csh
# @ output = poe.stdout.%(jobid).%(stepid)
# @ error = poe.stderr.%(jobid).%(stepid)
# @ network.MPI = csss,not_shared,us
# @ environment = MP_EUILIB=us; MP_EAGER_LIMIT=0
# @ node_usage = not_shared
# @ checkpoint = no
# @ ja_report = yes
# @ wall_clock_limit = 3800
# @ class = csl_reg
# @ job_type = parallel
# @ job_name = test.a1
## @ account_no = 00000000
# @ task_geometry = {(0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11,12,13,14)(15,16,17,18)
(19,20,21,22)(23,24,25,26)(27,28,29,30)(31,32,33,34)(35,36,37,38)(39,40,41,42)
(43,44,45,46)(47,48,49,50)(51,52,53,54)(55,56,57,58)(59,60,61,62)(63,64,65,66)(67)}
# @ queue
# ----- NCAR SGI O2K: chinook -----
# QSUB -q ded_28 -l mpp_p=30 # select batch queue
# SUB -lT 59:00 -lt 59:00 # set CPU time limits
# QSUB -mb -me -eo # combine stderr & stout
# ----- NCAR Compaq: prospect -----
#PBS -q reg
#PBS -l nodes=3:ppn=4
#PBS -l walltime=0:59:00
# ----- NOAA FSL Linux: jet -----
##$ -S /bin/tcsh
#$ -N run-pc_mpi
#$ -cwd
#$ -A crotc
#$ -pe comp 16
#$ -r y
#$ -l h_rt=02:00:00
#-----

```

The CCSM is supported on two platforms, the IBM SP and the SGI Origin 2000. Typically, the CCSM is run via a batch queuing system. The commands in the default script define the settings for three different batch queue environments at NCAR. The batch queuing system is machine- and site-dependent.

The `#!/bin/csh -f` line indicates that this is a C-shell script. The `-f` option keeps the users personalized `$HOME/.cshrc` file from being executed to avoid introducing aliases that could adversely

affect the operation of this script. The CVS lines document the revision control version of this script. The remaining lines in this section control the batch submission environment for the three different platforms. Refer to the relevant system documents for more information on these options.

On the IBM-SP, the task-geometry settings define how the model is to be used across the multi-processor nodes. This example (with only one # sign) is for the NTASK and NTHRD settings corresponding to the test.a1 T42\_gx1v3 resolution, set in section c of the script. The T31\_gx3v4 resolution in the paleoclimate test.a2 script use the smaller processor set defined in the commented-out (with two # signs) task\_geometry line above.

### 3.1.2 Define the common run environment

```

echo -----
echo b1. Set case sensitive environment variables available to model setup scripts
echo -----

setenv CASE      test.a1                # case name
setenv GRID      T42_gx1v3              # T42_gx1v3, T31_gx3v4, T62_gx1v3, T62_gx3v4
setenv RUNTYPE   startup                # startup, continue, branch, hybrid
setenv SETBLD    auto                   # auto, true, false
setenv BASEDATE  0001-01-01             # initial start date yyyy-mm-dd
setenv CASESTR   "fully coupled $GRID test" # short descriptive text string

```

In this section, the common run environment is defined. All of the components will share this environment. The variables defined in this section are:



<b>CASE</b>	(string) is the name that identifies this model run. The CASE name is propagated throughout the CCSM environment. It is used to define where the model run scripts are located, the area where the model is actually run and is used as part of the output file path name. Currently CASE can be up to 16 characters long.
<b>GRID</b>	(string) specifies the CCSM horizontal grid. The format is atm/lnd_locn/ice, where atm/lnd is resolution of the atmosphere and land components and ocn/ice is the resolution of the ocean and sea-ice components. the Currently distributed grids are T42_gx1v3, T31_gx3v4 for all configurations other than latm, and T62_gx1v3 or T62_gx3v4 for latm.
<b>RUNTYPE</b>	(string) specifies the state in which the CCSM is to begin a run. A startup run begins a new CCSM run from conditions that might involve reading data from external files or initializing variables internally or some combination. A hybrid run indicates the start of a new CCSM run largely from existing CCSM restart or initial files. A continuation run extends an existing CCSM run from its current endpoint, guaranteeing exact restart. A branch run defines a new CCSM run that is started from bit-for-bit exact restart files but with a new case name.
<b>SETBLD</b>	(string) controls whether or not the model executable is built. If <b>SETBLD = true</b> , all of the CCSM component executables will be rebuilt if gmake determines it is needed. For <b>SETBLD = false</b> , the component executables will not be rebuilt. For <b>SETBLD = auto</b> the components will not be rebuilt if <b>RUNTYPE</b> is continue. For all other <b>RUNTYPE</b> parameters, <b>SETBLD = auto</b> will invoke gmake. This ensures that the same component executables are used during the entire integration when running production runs using <b>RUNTYPE = continue</b> .
<b>BASEDATE</b>	(integer) defines the baseline date for this run. BASEDATE conforms to ISO-8601 format YYYY-MM-DD, where YYYY is the year in the Gregorian calendar, MM is the month of the year ranging from 01 for January to 12 for December and DD is the day of the month from 01 to 31.

```

setenv CSMDATA      /fs/cgd/csm/inputdata      # base dir for input data
setenv CSMROOT      /fs/cgd/home0/$LOGNAME/ccsm2.0 # root directory of source
setenv EXEROOT      /ptmp/$LOGNAME/$CASE      # run model here
setenv ARCROOT      /ptmp/$LOGNAME/archive/$CASE # archive root directory

```

<b>CSMDATA</b>	(string) is the root directory for the input and boundary datasets. This directory differs from the others in that it is not created by the CCSM run script. It is assumed that \$CSMDATA already exists and contains the data files in the CCSM2.0.1 input data distribution tar files downloaded from the CCSM2.0.1 release home page ( <a href="http://www.cesm.ucar.edu/models/ccsm2.0.1">www.cesm.ucar.edu/models/ccsm2.0.1</a> ).
<b>CSMROOT</b>	(string) defines the root directory of the CCSM code distribution directory tree. The model source code, scripts and documentation are located underneath this directory. In the default case, all of the following environment variables are derived from CSMROOT.
<b>EXEROOT</b>	(string) is the directory where the model actually executes. Subdirectories for each of the model components will be created under EXEROOT. These directories will contain the input and output datasets, the namelists, the model executables and other run artifacts.
<b>ARCROOT</b>	(string) defines a directory to be used by the CCSM tools when running the data harvesting scripts.

```
setenv REFCASE      test.a1          # Runtype=branch data case
setenv REFDATE     0001-01-06      # Runtype=branch start date
```

**REFCASE** (string) is the common reference case to use when starting up with a **RUNTYPE of branch**. This coordinates the CASE from which the CCSM will be branching across all of the components. **REFCASE** is ignored unless **RUNTYPE is set to branch**.

**REFDATE** (string) coordinates the date in REFCASE from which the branch run is to begin. **REFDATE** is ignored unless **RUNTYPE is set to branch**.

```
echo -----
echo b2. Select multi-processing and resolution specs
echo   The task and thread settings depend on the grid being used
echo   Use NTASK=1 and NTHR=1 for data models
echo -----

set MODELS = ( atm  lnd  ice  ocn  cpl ) # generic model names.
set SETUPS = ( atm  lnd  ice  ocn  cpl ) # setup script name
set NTASKS = (  8   3   16  40   1 )
set NTHRDS = (  4   4   1   1   4 )
```

This section defines the arrays of model components and their threading and tasking layouts.

The MODELS array defines the generic name of the model components to be coupled. Unless new components are being added, there should be no reason to change these settings. For each MODEL array element, a corresponding element definition is expected in the \$SETUPS, \$NTASKS and \$NTHRDS arrays.

The SETUPS array defines the specific names of the model components. These should align with the ordering on the \$MODELS array. The names set here will be used to identify which setup scripts

(i.e. \$SCRIPTS/\$SETUPS.setup.csh) will be called to build the individual model components. In this example, the \$SCRIPTS/atm.setup.csh will be called to build the atmosphere. If the data atmosphere is to be used instead of the active atmosphere model, "datm" should be used as the first element in the SETUPS array.

The NTASKS array sets the number of MPI tasks to be used for each model component.

The NTHRDS array sets the number of OPENMP threads to be used for each MPI task.

The example configuration is setup to execute on an IBM SP with 4 processors per node.

```

echo -----
echo c. The following environment variables can be set by the user but
echo   by default are derived from the environment variables above
echo -----

setenv MSSNAME 'echo $LOGNAME | tr '[a-z]' '[A-Z]'' # LOGNAME in caps

setenv MSSDIR mss:/$MSSNAME/csm/$CASE # MSS directory path name
setenv MSSDIR null:/dev/nul # MSS directory path name
setenv MSSRPD 0 # MSS file retention period
setenv MSSPWD $LOGNAME # MSS file write password

```

**MSSNAME** (string) follows a convention used by the NCAR Mass Store System. The first element in a user's Mass Store directory path is the user's login name in capital letters. This may be used by each of the components.

**MSSDIR** (string) defines the destination of the output datasets. mss:/PATH/name indicates that the datasets should be written to the NCAR mass store. WARNING: Some components (i.e. ice and ocn) do not obey this directive!!! cp:/file/path indicates that the datasets are to be copied into the directory /file/path. null:/dev/nul means do nothing. In this case, the first MSSDI example is overridden by the second, null, example. The output history files will be left in the directory where they were created. The archiver and harvester scripts in the \$SCRIPTS directory (see below) can be uncommented and used to send the output files to their final destination.

**MSSRPD** (integer) sets the NCAR Mass Storage System's retention period in days. The atmosphere and land models interpret a 0 MSSRPD value to mean that output data files will not be copied to the Mass Storage System.

**MSSPWD** (string) sets the NCAR Mass Storage System's write password.

```

setenv SCRIPTS $CSMROOT/scripts/$CASE # run scripts are here
setenv TOOLS $CSMROOT/scripts/tools # some tools are here
setenv LOGDIR $CSMROOT/scripts/$CASE/logs # save stdout here
setenv CSMCODE $CSMROOT/models # base dir for src code
setenv CSMUTL $CSMROOT/models/utills # Util directory
setenv CSMSHR $CSMROOT/models/csm_share # shared code dir
setenv CSMBLD $CSMROOT/models/bld # makefiles are here
setenv LID "'date +%y%m%d-%H%M%S'" # time-stamp/file-ID string

```

<b>SCRIPTS</b>	(string) is the directory containing the run scripts for the current CASE.
<b>TOOLS</b>	(string) is the directory containing CCSM utility tools.
<b>LOGDIR</b>	(string) is the directory to which copies of the standard out log files (print-out) from each of the component models will be copied.
<b>CSMCODE</b>	(string) points to the root directory of the CCSM source code for all components.
<b>CSMUTL</b>	(string) is the directory containing utility codes, such as the Earth System Modeling Framework (ESMF) routines.
<b>CSMSHR</b>	(string) is the directory holding CCSM code that is shared across a number of different components, such as timers, orbital settings, physical constants and message-passing wrappers.
<b>CSMBLD</b>	(string) is the directory containing the makefiles and site-specific gnumake macros necessary to build the model executables.
<b>LID</b>	(string) defines a unique time-stamp string of the form YYMMDD-hhmmss that is incorporated into the filenames of all of the component output files of the current run.

```
setenv OBJROOT $EXEROOT           # build code here
setenv LIBROOT $EXEROOT/lib       # Location of supplemental libraries
setenv INCROOT $LIBROOT/include   # Location of supplemental includes/modfiles
```

<b>OBJROOT</b>	(string) defines the directory where the model object files are to be created. While most systems allow OBJROOT and EXEROOT to be the same, some systems need these to be different.
<b>LIBROOT</b>	(string) is the directory where supplemental libraries (such as ESMF) will be built and maintained.
<b>INCROOT</b>	(string) is the directory for include and module files needed by the supplemental libraries.

```
setenv LFSINP $CSMDATA           # LOCAL INPUTDATA FSROOT
setenv LMSINP /CCSM/inputdata    # LOCAL INPUTDATA MSROOT
setenv LMSOUT /$MSSNAME/csm/$CASE # LOCAL OUTPUT MSROOT
setenv MACINP dataprocc.ar.edu   # REMOTE INPUTDATA MACHINE
setenv RFSINP /fs/cgd/csm/inputdata # REMOTE INPUTDATA FSROOT
setenv RMSINP /CCSM/inputdata    # REMOTE INPUTDATA MSROOT
setenv MACOUT dataprocc.ar.edu   # REMOTE OUTPUT MACHINE
setenv RFSOUT /ptmp/$LOGNAME/archive/$CASE # REMOTE OUTPUT FSROOT
```

These environment variables allow the user to configure variables to acquire input data and save output data.

<b>LFSINP</b>	(string) is the local file system disk location of the input data.
<b>LMSINP</b>	(string) is the root directory location on the local mass storage device (NCAR MSS or LANL/NERSC HPSS) of the input data.
<b>LMSINP</b>	(string) is the root directory on the local mass storage device (NCAR MSS or LANL/NERSC HPSS) for the output data.
<b>MACINP</b>	(string) is the remote machine to copy the input data from if the data cannot be located locally. The Unix scp command is used to transfer the files.
<b>RFSINP</b>	(string) is the remote file system directory on the MACINP machine for acquiring input data via scp.
<b>RMSINP</b>	(string) is the root directory on a remote mass storage device (NCAR MSS or LANL/NERSC HPSS) to acquire the input data from if those data cannot be located locally.
<b>MACOUT</b>	(string) is the remote machine to which the output data will be copied (via the Unix scp command).
<b>RFSOUT</b>	(string) is the remote file system directory on the MACOUT machine where the output data will be sent.

```

#--- logic to set BLDTYPE based on SETBLD above
setenv BLDTYPE $SETBLD
if ($SETBLD =~ auto*) then
  setenv BLDTYPE true
  if ($RUNTYPE == 'continue') setenv BLDTYPE false
endif
if ($BLDTYPE != 'true' && $BLDTYPE != 'false') then
  echo "error in BLDTYPE: $BLDTYPE"
  exit 1
endif

```

This logic resolves the **setenv SETBLD auto** option. For **setenv SETBLD auto**, gmake is run on the components only if RUNTYPE is set to startup, branch, or hybrid. If RUNTYPE is continue, component executables are assumed to already exist and rebuilds are not carried out. This ensures that the component executables are unchanged during the entire integration.

```

echo -----
echo d. Determine os/machine/site
echo -----

setenv OS unknown
setenv ARCH unknown
setenv MACH unknown
setenv SITE unknown

setenv OS `uname -s` # operating system
if ($status == 0) then # architecture
  if ( $OS == 'AIX') setenv ARCH IBM
  if ( $OS == 'OSF1') setenv ARCH CPQ
  if ( $OS == 'IRIX64') setenv ARCH SGI
  if ( $OS == 'Linux') setenv ARCH linux
endif

setenv MACHKEY `hostname`
if ($status == 0) then
  if ($MACHKEY =~ bb* ) setenv MACH babyblue # machine
  if ($MACHKEY =~ bf* ) setenv MACH blackforest
  if ($MACHKEY =~ s* ) setenv MACH seaborg
  if ($MACHKEY =~ prosp* ) setenv MACH prospect
  if ($MACHKEY =~ chin* ) setenv MACH chinook
  if ($MACHKEY =~ n* ) setenv MACH nirvana
  if ($MACHKEY =~ eag* ) setenv MACH eagle
  if ($MACHKEY =~ che* ) setenv MACH cheetah
  if ($MACHKEY =~ fal* ) setenv MACH falcon
  if ($MACHKEY =~ iam* ) setenv MACH lemieux
  if ($MACHKEY =~ g*.fsl.noaa.gov ) setenv MACH jet
  setenv SITE ncar # site, default is ncar
  if ($MACHKEY =~ n* ) setenv SITE lanl
  if ($MACHKEY =~ s* ) setenv SITE nersc
  if ($MACHKEY =~ eag* ) setenv SITE orn1
  if ($MACHKEY =~ che* ) setenv SITE orn1
  if ($MACHKEY =~ fal* ) setenv SITE orn1
  if ($MACHKEY =~ iam* ) setenv SITE psc
  if ($MACHKEY =~ g*.fsl.noaa.gov ) setenv MACH fsl
endif

```

This section tries to identify the site where the CCSM is being run.

<b>ARCH</b>	(string) returns the architecture of the machine on which the CCSM is being built.
<b>MACHKEY</b>	(string) returns the hostname of the machine on which the CCSM is being built.
<b>MACH</b>	(string) Converts \$MACHKEY into MACH for use in the scripts. If \$MACH is “unknown”, then the script will halt. \$MACH is used by the machine-specific files \$TOOLS/modules.\$OS.\$MACH and \$CSM-BLD/Macros.\$ARCH to build the CCSM.
<b>SITE</b>	(string) returns the computer center where the CCSM is being built based on \$MACHKEY.

```

echo -----
echo e. Create ccsm_joe
echo -----

setenv CSMJOE $SCRIPTS/ccsm_joe
rm -f $CSMJOE
$TOOLS/ccsm_checkenvs > $CSMJOE

```

The `ccsm_joe` file documents the job environment variables that are in effect for the run. This will aid in debugging any problems that might be experienced. `ccsm_joe` is also used by the data harvester and utility tools to get environment variables for the case.

The `ccsm_getrestart` utility positions restart files from the archive area. Use of this tool is commented out in the default version. This `ccsm` tools is a handy way to gather restart datasets from a central directory and copy them into the appropriate executable directories. This is often used when carrying out a branch or hybrid RUNTYPE and can be used for a continue RUNTYPE.

### 3.1.3 Select multi-processing and resolution specs

```

echo -----
echo f. Prepare $GRID component models for execution
echo   - create execution directories for atm,cpl,lnd,ice,ocn
echo   - invoke component model setup scripts found in $SCRIPTS
echo -----

setenv ATM_GRID 'echo $GRID | sed s/_.*//'; setenv LND_GRID  $ATM_GRID
setenv OCN_GRID 'echo $GRID | sed s/_.*//'; setenv ICE_GRID  $OCN_GRID

```

This section obtains grid information for use in the component setup scripts.

ATM\_GRID is set to the first part of \$GRID for use in the `atm.setup.csh` and `lnd.setup.csh` scripts.

OCN\_GRID is set to the second part of \$GRID for use in the `ocn.setup.csh` and `ice.setup.csh` scripts.

### 3.1.4 Run the setup script for each component

This section compiles and builds the CCSM component executables. In addition, many of the architecture dependent environment variables are set in this section.

```

#--- create working directories
foreach DIR ( $EXEROOT $LIBROOT $INCROOT $OBJROOT $LOGDIR)
  if !(-d $DIR) mkdir -p $DIR
end

#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

```

The **foreach DIR** loop creates the directories in the DIR list.

`$TOOLS/modules/$OS.$MACH` contains site specific module and environment settings. The modules at various sites do change with time, so if problems are encountered with compiling or linking in message passing libraries, the modules settings should be examined.

```

#--- create env variables for use in components
foreach n (1 2 3 4 5)
  set model = $MODELS[$n]
  setenv ${model}_dir $EXEROOT/$model; setenv ${model}_setup $SETUPS[$n]
  setenv ${model}_in $model.stdin ; setenv ${model}_out $model.log.$LID
end

#--- get restart files
#$TOOLS/ccsm_getrestart

```

This loop pre-defines environment variables for the run directory, the setup script, as well as the standard input and standard output file names.

```

echo -----
echo g. Build Earth System Modeling Framework   http://www.esmf.ucar.edu
echo -----

setenv EXEDIR $EXEROOT/esmf ; if !(-d $EXEDIR) mkdir -p $EXEDIR
cd $EXEDIR
echo `date` $EXEDIR/esmf.log.$LID | tee esmf.log.$LID
$SCRIPTS/esmf.setup.csh >>& esmf.log.$LID || exit 1

```

Various components of the CCSM use the Earth System Modeling Framework (ESMF) utilities. In this step the ESMF package is build, with the output from the build process being recorded in a log file. The ESMF documentation can be accessed from the URL shown above.



```

#--- create model directories for each platform
#--- ($EXEROOT/all for SGI, poe.cmdfile for AIX, prun.cmdfile for OSF1)
if (-e $EXEROOT/all) rm -rf $EXEROOT/all
mkdir -p $EXEROOT/all
echo "#! /bin/csh -f" >! $EXEROOT/prun.cmdfile

#--- initialize processor count
@ PROC = 0

```

The SGI directory “all” is cleared (used only on the SGI) and the prun.cmdfile file for the compaq and and the processor count variable , PROC, are initialized.

```

echo -----
echo h. Execute component setup.csh scripts, build models
echo -----
foreach n (1 2 3 4 5)
#--- activate stdin/stdout redirect work-around ---
#--- setup env variables for components and grids ---
  setenv MODEL $MODELS[$n] ; setenv SETUP $SETUPS[$n]
  setenv NTHRD $NTHRDS[$n] ; setenv NTASK $NTASKS[$n]
  setenv OBJDIR $OBJROOT/$MODEL/obj ; if !(-d $OBJDIR) mkdir -p $OBJDIR
  setenv EXEDIR $EXEROOT/$MODEL ; if !(-d $EXEDIR) mkdir -p $EXEDIR
  setenv THREAD FALSE ; if ($NTHRD > 1) setenv THREAD TRUE

```

The foreach loop cycles through the five-element arrays defined above. Each cycle through the loop will run the setup script for the MODELS component corresponding to the value of n (FORTRAN ordering).

First, a number of environment variables are defined identifying the specific component to be built (\$MODEL) and the setup script name (\$SETUP) that will be run to build the component. Next, the number of OMP threads (\$NTHRD) and number of MPI tasks associated with that component (\$NTASK) are resolved.

Names for the model execution (\$EXEDIR) and object (\$OBJDIR) directories are defined and these directories are created.

Finally, a true/false flag for OMP threading (\$THREAD) is set based on the value of \$NTHRD.

```

cd $EXEDIR
echo `date` $EXEDIR/$MODEL.log.$LID | tee $MODEL.log.$LID
$SCRIPTS/$SETUP.setup.csh >>& $MODEL.log.$LID
if ($status != 0) then
  echo ERROR: $MODEL.setup.csh failed, see $MODEL.log.$LID
  echo ERROR: cat $cwd/$MODEL.log.$LID
  exit 99
endif

#--- create model directories and processor counts for each platform
#--- ($EXEROOT/all for SGI, poe.cmdfile for AIX, prun.cmdfile for OSF1)
if ($n == 1) then
  rm -rf $EXEROOT/poe.cmdfile $EXEROOT/all; mkdir -p $EXEROOT/all
  echo "#! /bin/csh -f" >! $EXEROOT/prun.cmdfile
  @ PROC = 0
  if ($BLDTYPE == 'true') then
    cd $EXEROOT
    tar -cf $EXEROOT/$CASE.exe.$LID.tar $MODEL/$MODEL
  endif
else
  if ($BLDTYPE == 'true') then
    cd $EXEROOT
    tar -rf $EXEROOT/$CASE.exe.$LID.tar $MODEL/$MODEL
  endif
endif
endif

```

In the first time through the component build loop, a number of utility files, directories and counters are initialized. To keep the run script simple, all of these items are created whether they are needed or not.

On the SGI, mpirun requires that all of the model component executables exist in the same directory. If this directory, \$EXEROOT/all, exists, it is first deleted, then recreated.

On the Compaq systems, the file \$EXEROOT/prun.cmdfile will be created listing the model executables. Here, the first line of this file is created. The rest of the file will be made further on in this script.

The variable \$PROC is initialized. \$PROC will sum to the total number of processors requested.

If \$BLDTYPE == 'true', the component executable is added to the executable tar file. The executable tar file holds the copies of the component executables which are be used for this run and any subsequent runs where \$BLDTYPE == 'false'.

```

@ M = 0
while ( $M < $NTASK )
  echo "env OMP_NUM_THREADS=$NTHRD $MODEL/$MODEL" >>! $EXEROOT/poe.cmdfile
  echo "if ("\"$RMS_RANK == $PROC) ./ $MODEL/$MODEL" >>! $EXEROOT/prun.cmdfile;
  @ M++ ; @ PROC++
end
ln -s $EXEROOT/$MODEL/$MODEL $EXEROOT/all/.
end

```

Some machine specific bookkeeping is attended to here. The IBM SP and the Compaq machines require text files identifying the names of the component executables to be run under MPI. The counters for the number of tasks (\$M) and the processors (\$PROC) are incremented. The SGI O2K requires that all of the executable be run from a single directory, hence the link of all model executables in the to all/ directory. These constraints are handled here, the text files are created and the component executables are linked into a common directory. Again, to keep the run script simple, all of these items are created whether they are needed or not.

```
#-- save the latest executables to the active exe.tar
if ($BLDTYPE == 'true') then
  rm -f $EXEROOT/$CASE.exe.tar
  cp $EXEROOT/$CASE.exe.$LID.tar $EXEROOT/$CASE.exe.tar
endif
```

Finally, if \$BLDTYPE == 'true', the executable tar file for this build is made to be the default set of executables for this CASE.

### 3.1.5 Run the CCSM integration

The various supported platforms each have different environment settings that need to be specified to achieve optimum performance. Once these are set, the model is executed.

```
echo -----
echo i. Setup hardware specific env variables
echo -----

cd $EXEROOT
chmod 755 $EXEROOT/prun.cmdfile

if ( $OS == 'AIX' ) then
  limit datasize unlimited      ; setenv XLSMPOPTS "stack=86000000"
  setenv MP_EUILIB us           ; setenv MP_RMPPOOL 1
  setenv MP_NODES $PROC        ; setenv MP_PROCS $PROC
  setenv MP_PGMMODEL mpmd      ; setenv MP_CMDFILE      poe.cmdfile
  setenv MP_STDOUTMODE ordered ; setenv MP_SHARED_MEMORY yes
  setenv MP_EAGER_LIMIT 65536  ; setenv MP_INFOLEVEL      6
else if ( $OS == 'IRIX64' ) then
  setenv TRAP_FPE "UNDERFL=FLUSH_ZERO; OVERFL=ABORT,TRACE; DIVZERO=ABORT,TRACE"
  setenv OMP_DYNAMIC FALSE      ; setenv MPC_GANG OFF; setenv _DSM_WAIT SPIN
  unsetenv _DSM_VERBOSE         ; setenv _DSM_PLACEMENT ROUND_ROBIN
else if ( $OS == 'Linux' ) then
  source /usr/local/bin/setup-mpi.csh
  echo -n "NODEFILE:"
  cat $SGE_NODEFILE | awk '{printf("%s ", $1);}END{printf("\n");}'
  cd $EXEROOT/all
endif
env | egrep '(MP_|LOADL|XLS|FPE|DSM|OMP|MPC)' # document above env vars
```

Settings for the IBM SP are:

<b>limit datasize</b>	<b>unlimited</b> maximized the virtual memory allocation.
<b>XLSMPOPTS</b>	<b>"stack=8600000"</b> reserves 86 Mbytes of stack space for each thread.
<b>MP_EUILIB</b>	<b>us</b> requests User Space protocol for communications. This boosts performance for production runs by prohibiting other users from using the nodes where the model is running.
<b>MP_RMPOOL</b>	<b>1</b> tells POE to allocate nodes from resource manager pool 1.
<b>MP_NODES</b>	<b>\$PROC</b> sets the number of nodes over which the parallel tasks will be run.
<b>MP_PROCS</b>	<b>\$PROC</b> is the total number of processes for the model.
<b>MP_PGMMODEL</b>	<b>mpmd</b> identifies the programming model to be MPMD (Multiple Processes, Multiple Datastreams).
<b>MP_CMDFILE</b>	<b>poe.cmdfile</b> names the text file specifying the names of the component executables to be run under MPI.
<b>MP_STDOUTMODE</b>	<b>ordered</b> asks that standard out be buffered and flushed in the order of the tasks that wrote to standard out.
<b>MP_SHARED_MEMORY</b>	<b>yes</b> requests that all tasks running on the same node use shared memory for message passing on that node rather than communicating across the switch.
<b>MP_EAGER_LIMIT</b>	<b>65536</b> maximizes the message size of the receive data buffer for optimal performance.
<b>MP_INFOLEVEL</b>	<b>6</b> requests that all available informational messages be written to standard output.

Environment settings for the SGI Origin 2000 are:

```
setenv TRAP_FPE "UNDERFL=FLUSH_ZERO; OVERFL=ABORT,TRACE;
DIVZERO=ABORT,TRACE"
```

	traps floating point errors by setting floating-point values to zero when they become too small to represent or aborting on overflow or divide-by-zero.
<b>OMP_DYNAMIC</b>	<b>FALSE</b> forbids the use of dynamic scheduling for OpenMP threads.
<b>MPC_GANG</b>	<b>OFF</b> disallows "gang scheduling" to achieve higher performance with the OMP_DYNAMIC FALSE setting.
<b>_DSM_WAIT</b>	<b>SPIN</b> instructs each thread to wait in a loop without giving up the CPU until a synchronization event such as a lock or barrier succeeds.
<b>_DSM_VERBOSE</b>	requests that all available informational messages be written to standard output. While this is unset by default, <b>_DSM_VERBOSE</b> can be set to collect additional debugging information.
<b>_DSM_PLACEMENT</b>	<b>ROUND_ROBIN</b> specifies round-robin memory allocation for stack, data, and text.

```

echo -----
echo j. Run the model, execute models simultaneously allocating CPUs
echo -----

#exit          # UNCOMMENT to EXIT HERE, BUILD ONLY

echo "'date' -- CSM EXECUTION BEGINS HERE"
if ( $OS == 'AIX')    timex poe
if ( $OS == 'OSF1')  prun  -n $PROC csh -c prun.cmdfile
if ( $OS == 'IRIX64') mpirun -v -d $EXEROOT/all          \
    -np $NTASKS[1] "env OMP_NUM_THREADS=$NTHRDS[1] $MODELS[1]" : \
    -np $NTASKS[2] "env OMP_NUM_THREADS=$NTHRDS[2] $MODELS[2]" : \
    -np $NTASKS[3] "env OMP_NUM_THREADS=$NTHRDS[3] $MODELS[3]" : \
    -np $NTASKS[4] "env OMP_NUM_THREADS=$NTHRDS[4] $MODELS[4]" : \
    -np $NTASKS[5] "env OMP_NUM_THREADS=$NTHRDS[5] $MODELS[5]" &
if ( $OS == 'Linux') time /home/tierney/multi/mpirun --tombstone \
    -np $NTASKS[1] ./ $MODELS[1] : \
    -np $NTASKS[2] ./ $MODELS[2] : \
    -np $NTASKS[3] ./ $MODELS[3] : \
    -np $NTASKS[4] ./ $MODELS[4] : \
    -np $NTASKS[5] ./ $MODELS[5] &
wait
echo "'date' -- CSM EXECUTION HAS FINISHED"

```

Finally, the CCSM is run. On the IBM SP (`$ARCH == 'AIX'`), the Parallel Operating Environment (POE) is invoked. The information for the model configuration to run is input through the file specified with the `$MP_CMDFILE` environment variable. On the Compaq (`$ARCH == 'OSF1'`), the `prun` command executes the files listed in the `prun.cmdfile`. On the SGI, (`$ARCH == 'IRIX64'`), `mpirun` is called with the parallel tasking and threading information for each component being specified.

The `wait` command suspends the execution of the `test.a1.run` script until all background processes are complete.

### 3.1.6 Archive and harvest

In this step, the printed logs are archived and the output datasets are harvested.

```

echo -----
echo k. save model output stdout & stderr to $LOGDIR
echo   archive and submit harvester
echo -----

cd $EXEROOT
if (! -d $LOGDIR) mkdir -p $LOGDIR
gzip */*.$LID
if ($LOGDIR != "" ) cp -p */*.$LID.* $LOGDIR
#$SCRIPTS/ccsm_archive

```

Once the model has finished executing, the model standard output files are compressed and copied

to \$LOGDIR. If desired, the C-shell comment symbol, #, can be removed from the last line to run the csm\_archive tool script to archive the log file.

```

if ($OS == 'AIX') then
  set num = `llq | grep -i $LOGNAME | grep -i share | wc -l`
  cd $SCRIPTS
# if ($num < 1) llsubmit $CASE.har
endif
#if ($OS != 'AIX')      qsub $SCRIPTS/$CASE.har

```

A data harvester script (\$SCRIPTS/\$CASE.har) is used to transfer CCSM output data from the execution directories to a long-term storage device. Separating the harvesting function from the model execution allows model execution to continue even if the connections to the storage device are temporarily interrupted. **By default, the harvester is turned off** and all the output data will accumulate in the components' execution directories. Removing the C-shell comment symbol, #, will submit the harvester script for this case to the batch queue.

### 3.1.7 Resubmit

```

echo -----
echo 1. Resubmit another run script $CASE.run
echo -----

set echo
if ( -e $SCRIPTS/RESUBMIT ) then
  @ N = `cat $SCRIPTS/RESUBMIT`
  if ( $N > 0 ) then
    echo "Note: resubmitting run script $CASE.run"
    @ N--
    echo $N >! $SCRIPTS/RESUBMIT
    cd $SCRIPTS
    if ($OS == 'AIX') llsubmit $CASE.run
    if ($OS != 'AIX') qsub $CASE.run
  endif
endif

echo =====
echo m. end of nqs shell script
echo =====

```

The test.a1.run script ends a test to see if the model should be automatically resubmitted to the batch queues. If the file \$SCRIPTS/RESUBMIT exists and contains a number greater than 0, the test.a1.run script will be resubmitted to the batch queues. Then the number in the \$SCRIPTS/RESUBMIT file is decremented and rewritten to the file.

**WARNING:** It should be noted that if \$CASE.run has a RUNTYPE setting of startup, hybrid or branch, then the model will uselessly repeat the run that was just made. To avoid this, set the value of the counter in the file RESUBMIT to 0 until full production has begun using \$RUNTYPE "continue".

## 3.2 Sample Component Setup Script: `cpl.setup.csh`

The CCSM is designed to allow new component models to easily replace and existing component in the system. To encapsulate the different build procedures required by different component models, each CCSM component has a setup script designed to:

- Define the run environment of the individual component
- Position any input datasets needed by the component
- Build the component executable

In this section, the coupler setup script is used as an example of a typical component setup script. The component setup scripts, `$SCRIPTS/*setup.csh`, are called by `$SCRIPTS/test.a1.run`. Each component setup script prepares the component for execution by defining the run environment, positioning any restart or input datasets and building the component.

If the setup script is unable to complete any of these tasks, it will abort with a non-zero error status. The `test.a1.run` script checks the error status and will halt if an error is detected.

### 3.2.1 Document the setup script

```
#!/bin/csh -f
#=====
# CVS $Id: sample.setup.csh.tex,v 1.8 2002/10/10 19:17:03 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/sample.setup.csh.tex,v $
# CVS $Name: $
#=====
# cpl.setup.csh: Preparing a CSM coupler, cpl5, for execution
#
# (a) set environment variables, preposition input data files
# (b) create the namelist input file
# (b) build this component executable
#
# For help, see: http://www.cesm.ucar.edu/models/ccsm2.0.1
#=====
cat $0;$TOOLS/ccsm_checkenvs || exit -1          # cat this file, check envs
```

The first section typically documents the setup script.

The first line of this section identifies this as a C-shell script. The `"-f"` option prevents the user's personalized `$HOME/.cshrc` file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The cat command combines two functions on one line. The `"cat $0"` command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then `$TOOLS/ccsm_checkenvs` writes the environment variables that have been set by `test.a1.run` into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.

### 3.2.2 Set local component script variables

```
echo -----  
echo a. set environment variables, preposition input data files  
echo -----  
  
if ($GRID =~ T31* ) set ATM = ( T31 96 48 ) # name, x dimension, y dimension  
if ($GRID =~ T42* ) set ATM = ( T42 128 64 )  
if ($GRID =~ T62* ) set ATM = ( T62 192 94 )  
if ($GRID =~ T85* ) set ATM = ( T85 256 128 )  
if ($GRID =~ *gx3v4 ) set OCN = ( gx3v4 100 116 )  
if ($GRID =~ *gx1v3 ) set OCN = ( gx1v3 320 384 )  
  
if (!( $?ATM ) || !( $?OCN )) echo 'unknown GRID = ' $GRID  
if (!( $?ATM ) || !( $?OCN )) exit -1
```

Here the CCSM resolution variable is parsed into the atmosphere and ocean grid names and the number of points in the longitude and latitude directions are defined. If this is unsuccessful, the script aborts with a nonzero return status.



### 3.2.3 Define and position the input datasets

```

\rm -f map_*.nc

if      ($GRID == T31_gx3v4) then
  set MAP_A20F_FILE = map_T31_to_gx3v4_aave_da_020513.nc
  set MAP_A20S_FILE = map_T31_to_gx3v4_bilin_da_020513.nc
  set MAP_O2AF_FILE = map_gx3v4_to_T31_aave_da_020513.nc
  set MAP_R20_FILE  = map_r05_to_gx3v4_b6smooth_020613.nc
else if ($GRID == T31_gx1v3) then
  set MAP_A20F_FILE = map_T31_to_gx1v3_aave_da_020103.nc
  set MAP_A20S_FILE = map_T31_to_gx1v3_bilin_da_020103.nc
  set MAP_O2AF_FILE = map_gx1v3_to_T31_aave_da_020103.nc
  set MAP_R20_FILE  = map_r05_to_gx1v3_roff_smooth_010718.nc
else if ($GRID == T42_gx3v4) then
  set MAP_A20F_FILE = map_T42_to_gx3v4_aave_da_020807.nc
  set MAP_A20S_FILE = map_T42_to_gx3v4_bilin_da_020807.nc
  set MAP_O2AF_FILE = map_gx3v4_to_T42_aave_da_020807.nc
  set MAP_R20_FILE  = map_r05_to_gx3v4_b6smooth_020613.nc
else if ($GRID == T42_gx1v3) then
  set MAP_A20F_FILE = map_T42_to_gx1v3_aave_da_010709.nc
  set MAP_A20S_FILE = map_T42_to_gx1v3_bilin_da_010710.nc
  set MAP_O2AF_FILE = map_gx1v3_to_T42_aave_da_010709.nc
  set MAP_R20_FILE  = map_r05_to_gx1v3_roff_smooth_010718.nc
else if ($GRID == T62_gx3v4) then
  set MAP_A20F_FILE = map_T62_to_gx3v4_aave_da_020801.nc
  set MAP_A20S_FILE = map_T62_to_gx3v4_bilin_da_020801.nc
  set MAP_O2AF_FILE = map_gx3v4_to_T62_aave_da_020801.nc
  set MAP_R20_FILE  = map_r05_to_gx3v4_b6smooth_020613.nc
else if ($GRID == T62_gx1v3) then
  set MAP_A20F_FILE = map_T62_to_gx1v3_aave_da_010806.nc
  set MAP_A20S_FILE = map_T62_to_gx1v3_bilin_da_010806.nc
  set MAP_O2AF_FILE = map_gx1v3_to_T62_aave_da_010806.nc
  set MAP_R20_FILE  = map_r05_to_gx1v3_roff_smooth_010718.nc
else if ($GRID == T85_gx1v3) then
  set MAP_A20F_FILE = map_T85_to_gx1v3_aave_da_020405.nc
  set MAP_A20S_FILE = map_T85_to_gx1v3_bilin_da_020405.nc
  set MAP_O2AF_FILE = map_gx1v3_to_T85_aave_da_020405.nc
  set MAP_R20_FILE  = map_r05_to_gx1v3_roff_smooth_010718.nc
else
  echo "Using unsupported configuration, no mapping files set"
  exit 1
endif

$TOOLS/ccsm_getinput cpl/cpl5/$MAP_A20F_FILE $MAP_A20F_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_A20S_FILE $MAP_A20S_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_O2AF_FILE $MAP_O2AF_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_R20_FILE  $MAP_R20_FILE  || exit 1

```

This section controls the acquisition of the mapping datasets needed for the coupler. In general,

each component requires a unique set of input data files. All input datasets are all uniquely named by a description and a six digit number which documents the creation date (format: yymmdd) of the file. While the hard-wiring of the filenames restricts the degree of automation, it ensures that the exact data that the user requests is input into the model.

A few of the tools from the \$TOOLS directory make their first appearance here. The utility \$TOOLS/ccsm\_getinput will attempt to copy datasets from the input data directory into the current working directory.

If a copy of the data file is unavailable, the script will abort.

```

set RUN_TYPE = $RUNTYPE
if ($RUNTYPE == startup) set RUN_TYPE = initial
if ($RUNTYPE == hybrid) set RUN_TYPE = initial

set BASEDATE_NUM = 'echo $BASEDATE | sed -e 's/-//g''
if ($RUNTYPE == branch) then
  set REST_BFILE = $REFCASE.cp15.r.${REFDATE}-00000
  echo set REST_BFILE = $REST_BFILE
  $TOOLS/ccsm_getfile $REFCASE/$MODEL/rest/${REST_BFILE} || exit 99
  set BASEDATE_NUM = 'echo $REFDATE | sed -e 's/-//g''
else
  set REST_BFILE = 'null'
endif

```

A number of common variables are defined in the \$SCRIPTS/test.a1.run. Individual CCSM components often need to translate the common variables into different names or formats that the component can read. Here, \$RUNTYPE, \$BASEDATE, \$REFCASE and \$REFDATE are evaluated for use by the coupler. The coupler recognizes both “startup” and “hybrid” runtypes as coupler “initial” runs. The coupler needs a different date format (\$BASEDATE\_NUM) than supplied by \$BASEDATE. For “branch” runs, the coupler uses \$REFCASE and \$REFDATE to generate the branch filename and date.

## 3.2.4 Write the input namelist

```

echo -----
echo b. create the namelist input file
echo -----

cat >! $MODEL.stdin << EOF
  &inparm
  case_name   = '$CASE '
  case_desc   = '$CASE $CASESTR '
  rest_type   = '$RUN_TYPE '
  rest_date   = $BASEDATE_NUM
  rest_bfile  = '${REST_BFILE} '
  rest_pfile  = '$SCRIPTS/rpointer.$MODEL'
  map_a2of_fn = '$MAP_A2OF_FILE'
  map_a2os_fn = '$MAP_A2OS_FILE'
  map_o2af_fn = '$MAP_O2AF_FILE'
  map_r2o_fn  = '$MAP_R2O_FILE'
  stop_option = 'ndays'
  stop_n      = 5
  hist_freq   = 'ndays'
  hist_n      = 5
  rest_freq   = 'ndays'
  rest_n      = 5
  diag_freq   = 'ndays'
  diag_n      = 1
  info_bcheck = 2
  hist_tavg   = 1
  orb_year    = 1990
  flx_epbal   = 'off'
  flx_albav   = 0
  mss_dir     = '$MSSDIR/$MODEL/ '
  mss_rtpd    = $MSSRPD
  mss_pass    = '$MSSPWD'
  mss_rmlf    = 0
  nx_a = $ATM[2] , ny_a = $ATM[3] , nx_l = $ATM[2] , ny_l = $ATM[3]
  nx_o = $OCN[2] , ny_o = $OCN[3] , nx_i = $OCN[2] , ny_i = $OCN[3]
  /
EOF
echo o contents of $MODEL.stdin: ; cat $MODEL.stdin ; echo ' '

```

This section constructs the input namelist that is used to control runtime operation of the component. In the namelist input file, a wide range of predefined parameters are set to control the behavior of the component. The namelist input file, in this case called `cpl.stdin`, is a text file that is read by the component model. Namelist input for components consists of text strings enclosed in quotes, integer and real numerical values and logicals.

The "cat" command uses the C-shell here-document option to create the file `$EXEDIR/cpl.stdin` with all the settings being evaluated to the current values of the specified environment variables.

<b>&amp;inparm</b>	is the namelist group name, which matches the groupname defined within the coupler.
<b>case_name</b>	= <b>'\$CASE '</b> (string) sets a unique text string (16-characters or less) that is used to identify this run. The CASE variable is set in \$SCRIPTS/test.a1.run and is used extensively in the CCSM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX filenames characters such as letters, numbers, underscores, dashes, commas or periods.
<b>case_desc</b>	= <b>'\$CASE \$CASESTR '</b> (string) provides 80 characters to further describe this run. This description appears in the output logs and in the header data for the output data sets. CASESTR is set in the \$SCRIPTS/test.a1.run script.
<b>rest_type</b>	= <b>'\$RUN_TYPE '</b> (string) specifies the state in which the coupler is to begin the run. rest_type settings initial, branch and continue map into the CCSM variables startup or hybrid, branch and continue).
<b>rest_date</b>	= <b>\$BASEDATE_NUM</b> (string) is the initial date of the simulation. This variable is ignored on continuation or branch runs.
<b>rest_bfile</b>	= <b>'\$REST_BFILE'</b> (string) specifies the branch file to use when starting a branch run. This ignored unless rest_type is set to 'branch'.
<b>rest_pfile</b>	= <b>'\$SCRIPTS/rpointer.\$MODEL'</b> (string) is the complete filepath and filename of the restart "pointer file" used for continuation runs.
<b>map_a2of_fn</b>	= <b>'\$MAP_A2OF_FILE'</b> (string) is the filename of the map for atmosphere-to-ocean flux fields.
<b>map_a2os_fn</b>	= <b>'\$MAP_A2OS_FILE'</b> (string) is the filename of the map for atmosphere-to-ocean state fields.
<b>map_o2af_fn</b>	= <b>'\$MAP_O2AF_FILE'</b> (string) is the filename of the map for ocean-to-atmosphere flux fields.
<b>map_r2o_fn</b>	= <b>'\$MAP_R2O_FILE'</b> (string) is the filename of the map for land-runoff-to-ocean.
<b>rest_freq</b>	= <b>'monthly'</b> (string) instructs the coupler to have all the CCSM components write out restart files on the first day of every month.
<b>rest_n</b>	= <b>3</b> (integer) when rest_freq is set to 'nday', rest_n sets the number of days between writes of the restart files. Since rest_freq is 'monthly', this setting is ignored.
<b>diag_freq</b>	= <b>'ndays'</b> (string) sets the frequency at which diagnostics are printed from the coupler. In this case, the setting ndays will use the number of days set by diag_n.
<b>diag_n</b>	= <b>1</b> (integer) specifies the number of time periods for the time unit set in diag_freq.
<b>stop_option</b>	= <b>'ndays'</b> (string) controls the length of the CCSM run.
<b>stop_n</b>	= <b>5</b> (string) specifies that this integration will run for 5 days.
<b>hist_freq</b>	= <b>'monthly'</b> (string) controls the frequency of history file output.
<b>hist_n</b>	= <b>1</b> (integer) is the option when hist_freq = 'ndays' or 'nstep'. Since hist_freq is 'monthly' this setting is ignored.
<b>info_bcheck</b>	= <b>2</b> (string) specifies that high precision printed output is to be written every day into the coupler log file. This is used for verifying that two runs are exactly the same.
<b>orb_year</b>	= <b>1990</b> (integer) is the calendar year that is used to determine the solar orbit and resulting solar angles.

<b>flx_epbal</b>	= 'off' (string) turns off evaporation/precipitation balancing.
<b>flx_albav</b>	= 0 (integer) turns off daily average albedos.
<b>mss_dir</b>	= '\$MSSDIR/\$MODEL/' (string) sets the pathname of the NCAR Mass Storage System (MSS) files.
<b>mss_rtpd</b>	= \$MSSRPD (integer) sets the retention period when using the NCAR MSS.
<b>mss_pass</b>	= '\$MSSPWD' (string) sets the write password when using the NCAR MSS.
<b>mss_rmlf</b>	= 0 (integer) does not remove local files after mswrite.
<b>nx_a</b>	(integer) is the latitude dimensions of the atmosphere model.
<b>ny_a</b>	(integer) is the longitude dimensions of the atmosphere model.
<b>nx_l</b>	(integer) is the latitude dimensions of the land model.
<b>ny_l</b>	(integer) is the longitude dimensions of the land model.
<b>nx_o</b>	(integer) is the latitude dimensions of the ocean model.
<b>ny_o</b>	(integer) is the longitude dimensions of the ocean model.
<b>nx_i</b>	(integer) is the latitude dimensions of the sea-ice model.
<b>ny_i</b>	(integer) is the longitude dimensions of the sea-ice model.
<b>/</b>	marks the end of the inparm namelist group.
<b>EOF</b>	marks the end of the here document begun with the "cat" command

Detailed information on the coupler namelist variables can be found in the coupler User's Guide.

### 3.2.5 Build the component executable

```

echo -----
echo c. Build an executable in $OBJDIR
echo -----

cd $OBJDIR

# Filepath: List of source code directories (in order of importance).
#-----

\cat >! Filepath << EOF
$SCRIPTS/src.$MODEL
$CSMCODE/cpl/cpl5
$CSMSHR
EOF

# run make
#-----
#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c || exit 2
  gmake -j 6 VPFILE=Filepath MODEL=cpl5 EXEC=$EXEDIR/$MODEL \
    -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS || exit 2
else
  echo "BLDTYPE = $BLDTYPE"
endif

```

The CCSM uses the gnumake (also known as “gmake”) tool to build the model executable. Each of the components setup scripts creates a list of source code directories from which to gather the input source code for that component. This list is called Filepath and will be used as the input to the gmake VPATH list. The file Filepath is written in each of the components \$OBJDIR directories.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.cpl, is typically used to hold modified coupler source code. If a directory in the Filepath list is either empty or doesn’t exist at all, no error will result. In general, the directories \$SCRIPTS/src.\$MODEL can be used to store locally modified source code. Each component script recognizes this directory as the top priority for finding source code.

First the makdep code is compiled. This utility program is called by the Makefile and checks for source code dependencies. This is done by seeing if any of the header or include files have been updated since the model was last built and ensures that the F90 modules are constructed in the proper order.

Once makdep is compiled, the GNU make program, gmake, is used to actually build the model. The -j option specifies the number of processors to use to build the model. The -f \$CSMBUILD/Makefile points to the generic CCSM Makefile while MACFILE=\$CSMBLD/Macros.\$OS points to the machine specific make options. MODEL identifies the component being built and VPFILE points to the Filepath list. Finally, the actual executable to be built is \$EXEDIR/\$MODEL.

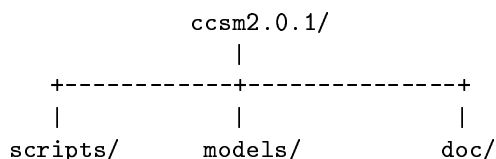
At this point, control is returned to test.a1.run.

## 4 Building the CCSM

CCSM2.0.1 is supported on IBM SP2 and SGI Origin 2000 platforms. These platforms represent examples of both distributed-memory and shared memory-architectures. A sample script is distributed with the model to illustrate how to configure the model to run on these different architectures, at different resolutions, with different combinations of models.

### 4.1 The CCSM directory structure

The CCSM2.0.1 code distribution consists of three directories:



In the **scripts/** directory are sample CCSM run scripts, tool scripts and a graphical user interface for constructing different CCSM configurations. The **scripts/test.a1/** directory contains the master run script (**scripts/test.a1/test.a1.run**) and all the setup scripts necessary to build and run the standard CCSM on the supported platforms. The **scripts/test.a2** directory has the same for the paleoclimate version.

The **scripts/test.a1/test.a1.run** script has three main tasks:

- 1 Define common build and run environment variables.
- 2 Run the setup script for each component.
  - a. Acquire any initial or boundary datasets needed by that component.
  - b. Generate the components namelist input file.
  - c. Build the component executable.
- 3 Execute all components simultaneously.

The common build and run environment variables set in the run script are automatically propagated to each of the component model setup scripts. The setup scripts will use tools found in the **scripts/test.a1/tools**. In **scripts/test.a1/gui\_run** is a graphical user interface to generate different CCSM configurations based on the test.a1 sample scripts.

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:

<b>models/bld/</b>	Gnumake Makefiles and machine-dependent macro files
<b>models/cpl/</b>	Flux Coupler source code
<b>models/atm/</b>	source code for all atmosphere models
<b>models/ice/</b>	source code for the sea-ice models
<b>models/ocn/</b>	source code for the ocean models
<b>models/land/</b>	source code for the land models
<b>models/utils/</b>	common utilities source code
<b>models/csm_share/</b>	source code shared by more than one CCSM component
<b>models/camclm_share/</b>	source code shared by CLM and CAM only

The input data necessary to run the model is distributed separately and is discussed below.

### 4.2 \$CSMBLD contents

The directory **\$CSMBLD** contains the files necessary to build the CCSM components.







## 5 Running the CCSM

This section will describe some of the practical aspects of setting up a short test run or a longer production run. The best way to get scripts configured for either a test or production run is to use the CCSM Graphical User Interface (GUI), described elsewhere. This section will provide some guidance about how to set up a configuration manually.

### 5.1 Start up

There are a number of steps required to configure a CCSM run manually.

#### 5.1.1 Running a simple test case of the fully coupled model on the NCAR IBM machine, blackforest

Assume that the new case name is mytest1 and the root directory where the CCSM model is located in `/home/$LOGNAME/ccsm2.0.1/`.

- Get a copy of the CCSM source code tarfile and untar it in the directory `/home/$LOGNAME`. Then create the a new case name for your experiment (in this example, it will be “mytest1”) and copy the contents of the test.a1 into the new case directory as follows:

```
cd /home/$LOGNAME/ccsm2.0.1/scripts
mkdir mytest1
cp test.a1/* mytest1/
cd /home/$LOGNAME/ccsm2.0.1/scripts/mytest1
mv test.a1.run mytest1.run
mv test.a1.har mytest1.har
```

- Modify the main run script by editing editing mytest1.run and making the following changes:

```
change "job_name" to mytest1
change "setenv CASE" to "mytest1"
change "setenv CASESTR" to a useful string
change "setenv CSMROOT" to /home/$LOGNAME/ccsm2.0.1
change "setenv CSMDATA" to the local path to the inputdata directory
change "setenv EXEROOT" to the local directory where the model will run
change "setenv ARCRROOT" to the local directory for archiving model output
```

- Run the script by submitting `/home/$LOGNAME/ccsm2.0.1/scripts/mytest1/mytest1.run` to the batch queue with following command:

```
llsubmit mytest1.run
```

#### 5.1.2 Changing the configuration

To modify the model configuration, several changes must be made to the main run script. Assume a model case name of “mytest2” for this case. Go through the section above using mytest2 as the case name. Then in addition to the changes in the section above, modify the main script, mytest2.run, as follows:

- Modify SETUPS:

- change "set SETUPS" to reflect the configuration you'd like
- atm can be set to atm, datm, or latm
- lnd can be set to lnd or dlnd
- ice can be set to ice or dice
- ocn can be set to ocn or docn

- Modify NTASKS/NTHRDS:

change "set NTASKS" and "set NTHRDS" to reflect the configuration used in SETUPS above. NTASKS represent the number of MPI tasks, and NTHRDS represents the number of OpenMP threads per MPI task.

The total number of processors used for each CCSM component is  $NTASKS * NTHRDS$ . The NTASKS and NTHRDS array elements are aligned consistently with the MODELS and SETUPS arrays.

- The data models datm, docn, dice, dlnd, and latm should have  $NTASKS=1$ ,  $NTHRDS=1$
- atm can run all MPI, OPENMP, or combined MPI/OPENMP
- lnd can run all MPI, OPENMP, or combined MPI/OPENMP
- ice runs MPI only, therefore NTHRDS must be set to 1
- ocn runs MPI only, therefore NTHRDS must be set to 1
- cpl runs OPENMP only, therefore NTASKS must be set to 1

- Modify the batch queue information to be consistent with the configuration specified by NTASKS and NTHRDS:

- For the IBM, this is "task\_geometry"
- For the SGI, this is on the "QSUB -l" line
- For the CPQ, this is on the "PBS nodes/ppn" line

As an aside, there is a standard shorthand naming convention for configuration setups. In particular,

- A = datm,dlnd,docn,dice,cpl
- B = atm,lnd,ocn,ice,cpl
- C = datm,dlnd,ocn,dice,cpl
- D = datm,dlnd,docn,ice,cpl
- F = atm,lnd,docn,ice (prescribed ice mode),cpl
- G = latm,dlnd,ocn,ice,cpl
- H = atm,dlnd,docn,dice,cpl
- I = datm,lnd,docn,dice,cpl
- K = atm,lnd,docn,dice,cpl
- M = latm,dlnd,docn,ice (mixed layer ocean mode), cpl

This case-naming convention is used in the GUI, and the tested configurations are summarized in the section describing supported configurations.

### 5.1.3 Changing the RUNTYPE

RUNTYPE is set in the main run script and determines how the CCSM run is to be started. RUNTYPE can be "startup", "continue", "branch", or "hybrid".

<b>startup</b>	represents a new case started from some model specific initial files or state.
<b>continue</b>	is a continuation of a case and guarantees exact restart capability.
<b>branch</b>	is like a continuation run, but the CASE name is changed. A set of restart files is used to start a branch run and exact restart is guaranteed if source code or model input hasn't changed. Typically, however, the purpose of a branch run is the evaluate the impact of a modification of the model. The exact restart guarantee ensures that any differences between the original run and the branch run are due to the modification introduced to the branch run.
<b>hybrid</b>	is a startup from atmosphere and land initial condition files and ocean and ice restart files. The model is started as if it were a startup case with a 1 day lag in the start of the ocean model. An exact restart is not guaranteed due to the atmosphere and land models using "initial" files and because of the ocean time lag.

For startup and continue runs, no specific script changes are required. For continue runs, the appropriate restart files must be placed in the executable directories. The scripts attempt to do this for the case automatically by searching directories for restart files. For branch runs, the environment variables REFCASE and REFDATE must be set to the name of the previous case and date in the REFCASE run where the new branch run will started from. Those restart files must be available to the new case. For hybrid runs, the environment variables REFCASE, REFDATE, and BASEDATE must be set in the main script. Those represent the prior case and date and the new starting date for this case. Hybrid runs allow a change to both case and starting date.

The RUNTYPES startup, branch, and hybrid are all used to start a new case. The RUNTYPE continue is used to continue any run, no matter what initial RUNTYPE was used.

### 5.1.4 Running on a new machine

The CCSM release is targeted at the NCAR IBM SP. Due to the distinct nature of individual computer sites, many aspects of the CCSM scripts may need to be changed when running on a new machine. These include

- batch queue commands at the top of the main script
- values of environment variables OS, SITE, MACH, and ARCH in the main script
- the names of CCSM paths set by environment variables in the main script
- the check of the submission status of the harvester script near the end of the main script
- the harvester script
- interaction with the local mass storage system via the tools scripts ccsm\_msread, ccsm\_msmkdir, and ccsm\_mswrite
- machine dependent modifications to the models/bld/Macros.\* file

General guidance about specific aspects of the scripts that need to be changed in order to run on different machines can be found in the scripts/tools/test.a1.mods.\* files. The \* represents the hostname of various machines where the model has already been run. Each of the files contains the machine-specific modification that are required.

### 5.1.5 Getting into production

There is a specific procedure that should be used to start a production run. The model defaults are set to run a short test case. The general procedures for starting a production run are as follows.

- Setup a startup, branch, or hybrid case
- For branch or hybrid
  - mkdir \$ARCROOT/restart
  - place all required restart/initial files in \$ARCROOT/restart
  - make a backup copy of all these files, this directory will be deleted at the end of the first run:

```
mkdir $ARCROOT/restart.bu
cp $ARCROOT/restart/* $ARCROOT/restart.bu
```

- Activate (uncomment) the following lines in the main script:
  - # \$TOOLS/ccsm\_getrestart
  - # \$SCRIPTS/ccsm\_archive
- Modify the namelist variables in the scripts/\$CASE/cpl.setup.csh script to carry out a production length run. A good starting point is to set the following namelists to make a three month run:

```
diag_n      = 10
stop_option = 'nmonths'
stop_n      = 3
info_bcheck = 0
```

- Submit the first run
- Verify the first run completed as expected, including archiving
- Once the first run has completed, set RUNTYPE to "continue"
- Activate (uncomment) the following line in the main script to allow the submission of the harvester at the end of the run:

```
# if ($num < 1) llsubmit $CASE.har
```

- Optionally re-comment the ccsm\_getrestart line in the main script:

```
# $TOOLS/ccsm_getrestart
```

- Submit the continue run
- Verify the continue run completed as expected including archiving and harvesting
- Place a non-zero positive integer in the file scripts/\$CASE/RESUBMIT. The file RESUBMIT controls automatic resubmission. This integer counter in this file decrements each time the model is resubmitted until it gets to zero, at which point the model will not resubmit itself:

```
echo 10 >! \${SCRIPTS}/RESUBMIT
```

- Resubmit the continue run
- Monitor the run including archiving and harvesting

## 5.2 What is the `ccsm_joe` file?

The `ccsm_joe` (Job Operating Environment) file is created by the main CCSM run script every time it executes. It can be thought of as a case specific resource file for CCSM. The `ccsm_joe` file contains a summary of some of the CCSM-specific environment variables. It is a useful debugging tool as it summarizes many of the important variables in the latest run.

It is also used by other scripts to determine the case specific variables. The CCSM harvester (`$CASE.har`), archiver (`ccsm_archive`), and a number of the scripts in the CCSM tools directory (`SCRIPTS/tools`) use this file to set case specific variables.

## 5.3 How does auto-RESUBMIT work?

The CCSM model will automatically resubmit the main run script if the integer parameter in the file `SCRIPTS/RESUBMIT` is greater than zero. In section (h) of the main run script, the script captures the value in the RESUBMIT file, decides whether to resubmit and if so, decrements the integer in the RESUBMIT file. When the resubmit parameter decrements to zero, auto resubmission will stop. This provides flexibility to the user to prevent runaway jobs. Initially, users should set the RESUBMIT integer to some moderate value, like 2. Once confidence has been established, the integer in RESUBMIT can be increased. The default value is zero, so the script will not RESUBMIT automatically by default.

When using RESUBMIT, RUNTYPE should usually be continue; otherwise the same initial period of the run will likely be run over and over.

### 5.3.1 Runaway jobs

Occasionally, the model will stop prematurely (due to a hardware problem or a model problem). If this happens, often the scripts will continue to resubmit themselves and this will usually lead to a “runaway” situation. To stop runaway jobs, first set the resubmit parameter to zero in the RESUBMIT file, then try to kill currently active runaway jobs.

## 5.4 Batch queuing challenges

Generally, each machine at each site has a unique batch queueing environment. This is less true for IBM machines which seem to use loadleveler nearly universally. Even with loadleveler, however, different sites have different loadleveler configurations. In particular, users may need to change the network parameter and class. Certainly with any queueing systems, users need to be aware of queue names, processor resources, and time limits.

CCSM has been run under loadleveler, NQS, LSF, and PBS on various machines. In the `$TOOLS` directory are files named `test.a1.mod.*`. These files provide guidance about both hardware and batch setups for specific machines.

The default values in the CCSM script provide batch commands for loadleveler, NQS, and PBS batch systems. The file `$TOOLS/test.a1.mods.nirvana` provides guidance for an LSF queueing system.

Users need to be careful to implement appropriate changes to the CCSM scripts for their particular environment.

**NOTE:** Both the main run script and the harvester script are set up to run in NCAR batch environments and both will need to be modified when using non-NCAR batch queueing systems.

## 5.5 Modifying source code

Source code is provided with the CCSM release. It is not unreasonable to make code changes directly in the files within the CCSM models directories and then rebuild and run the model.

However, the recommended approach is to create directories named `src.atm`, `src.lnd`, `src.ice`, `src.ocn`, and `src.cpl` in the directory where the case-specific scripts reside, `SCRIPTS/CASE`. Users can then copy source code files from the main CCSM models directories into these component-specific directories and then modify those files directly. By default, the CCSM scripts and build environment will use files in the `src.*` directories before files in the models source directories. In other words, source code in `src.*` directories has higher priority than source code in the models directories.

There are a number of benefits to doing this. First, it preserves the release source code so that differences can be carried out later and users can backtrack to the generic release source if desired. Second, it allows users to configure multiple experiments, each with each experiment having its own unique source code changes without requiring copies of the entire source tree. For instance, a sensitivity study could be carried out on a source code specified parameter by just copying one source code file into the `src.*` directory for a number of cases and then modifying that file in the `src.*` directory.

## 5.6 CCSM Data Management

This section briefly outlines the data flow for a production run. All binaries, model input, and model output exist at some point in specific directories in the executable area. The scripts generate the model binaries and get model input. The model then executes, and model output (restart files, history files, and log files) are written directly into the executable directories. Once the model has completed execution, the script `ccsm_archive` is run and the model output is moved into the archival directory. Subdirectories are created in the archive directory for each component as well as a restart and `restart.tars` directory. The most recent set of restart files, including pointer files, are copied into the restart subdirectory of the archive directory. That directory is tarred up and copied to the `restart.tars` directory.

Once model output has been archived, the harvester executes, checks in files in the archive directory. If the files have successfully been moved to the mass store previously, the copy of the file in the archive directory is removed. If that file has not been copied to the mass store previously, the harvester performs that operation. In effect, the harvester must pass through the model output files twice in the archive area before removing them. On the first pass, the file is copied to the mass store, and on the second pass the existence of the mass store copy is verified and the local copy of the file is removed.

## 5.7 What is harvesting doing?

The CCSM harvester is a separate job that saves the model output to a separate file storage device, generally known as a mass storage system. This could be the NCAR mass store, an `hpss`, or similar. The overall CCSM data flow is described separately. A part of the data flow is moving model output from the executable directories to a temporary archive directory by a script called `ccsm_archive`. The harvester is then used to move data from the archive directory to the mass storage system.

The harvester script can be found in the main `SCRIPTS/CASE` directory and is usually named `CASE.har`. The harvester can be run interactively or in batch mode. The harvester provided should be considered a template for customizing the harvesting process. Each user and each site may have different needs for harvesting. The harvesting script largely takes advantage of scripts in the `SCRIPTS/tools` directory.

Overall, the harvester works as follows:

- Loop through the archive directories for each component
- Loop through each file in the component archive directory

- Verify whether the file is already on the mass store and is bit-for-bit identical with the file sitting on disk. This is done by reading the file off the mass store into a local temporary filename, checkmss. Then the two files via the Unix compare command “cmp”
- If the local file and the mass store file are identical, remove the local file
- If the local file and the mass store file are different, write the local file to the mass store
- For some subset of files, copy the files to another area or another machine. This is done only if the local variable name, copyfiles\_to\_othersite, is true

The harvester takes advantage of the local copy of ccsm\_joe to determine many of the case specific variables. A number of harvester variables are set in the main run script including \$LMSOUT, \$MACOUT, and \$RFSOUT.

## 5.8 Monitoring the integration

Several steps must be taken to monitor the integration. These include monitoring the model run, archiving, harvesting, and disk quotas.

## 5.9 Data processing

All history output files are netCDF files and conform to the CF netCDF metadata convention. Many tools for processing, analyzing and visualizing netCDF files can be found on the netCDF web site: [www.unidata.ucar.edu/netcdf](http://www.unidata.ucar.edu/netcdf).

## 5.10 Comparing output to NCAR controls

NCAR control output is normally available on the NCAR mass storage system. Check the CCSM web page [www.cesm.ucar.edu](http://www.cesm.ucar.edu) for the latest information on model output availability.

# 6 Testing the CCSM

This section will describe some of the practical tests that can and should be run with CCSM to validate model changes. The section will be expanded in the future.

## 6.1 Exact restart test

Anyone making changes to the CCSM is strongly advised to do a short exact-restart test of the model. This involves making a 5 day startup/hybrid/branch run plus a 5 day continuation run and a 10 day startup/hybrid/branch run covering the same period. At the end of the 10 days, both runs must produce exactly the same answers. This can be verified by differencing the model restart files.



## 7 Common Aborts, Errors, Debugging, and Performance Issues

This section provides some suggestions about how to overcome those nagging problems that sometimes occur. There is also a short discussion on performance issues.

### 7.1 Common aborts and errors

Occasionally problems crop up when trying to build or run the CCSM system. This section will help users solve some of these problems. The most obvious places to look for error messages are in the model log files or in the batch standard out or standard error files. On some systems, these messages will be mailed to the user.

Many of the fixes require changes to parameter settings. All changes to CCSM settings made during a run should be documented to ensure the scientific reproducibility of your results.

#### 7.1.1 Component model has trouble building

Try making a clean start. Each component has a build directory called `$OBJROOT/$model/obj` (normally, this is equivalent to the `$EXEROOT/$model/obj` directory). To make a clean start, use the command: `rm -r -f $OBJROOT/*/obj`. This will remove the model object files and force the model to completely rebuild the next time. It is usually sufficient to only remove the `obj` directory for the component that seems to be causing problems.

When rebuilding the model, make sure the `$SETBLD` is set to true in the main run script if `$RUNTYPE` is set to continue. `$SETBLD` must be true or auto if `$RUNTYPE` is startup or hybrid or branch. Otherwise, the model will not build when submitted.

The main script can be run interactively on most platforms for build purposes. It may be necessary to kill the interactive script before the model starts running or activate the exit line that stops the script before the model begin executing.

If there is a source code problem, the compiler error will show up in the model log files. The main build script typically exits and indicates which model log file to interrogate.

#### 7.1.2 Model won't continue due to restart problem

If a restart file is corrupted, the coupled model likely will not run. First, verify that user quotas have not been exceeded and verify that the disk is not full. Then check if the size of the restart file in the `$EXEDIR` and the `$ARCROOT/restart` directory is different than the previous restart files written by that component. If either of these are problems, action must be taken before the model can be restarted.

If there is a corrupt restart file, remove the copy of the file in the executable directory. If the copy of the restart file in `$ARCROOT/restart` is also corrupt, remove that copy of the file. Check to see if there is a valid copy of the file in `$ARCROOT/$model` or on the local mass store. If so, copy that file into `$ARCROOT/restart` and try resubmitting the model. If there are no valid copies of a restart file, back up to the last valid restart set by removing all files in the `$ARCROOT/restart` directory and untarring the last valid restart dataset residing in `$ARCROOT/restart.tars`. Alternatively, delete all files in `$ARCROOT/restart`, manually gather a set of restart files from either `$ARCROOT/$model` or the local mass store into `$ARCROOT/restart`, copy the rpointer files from `$SCRIPTS` to `$ARCROOT/restart`, modify the rpointer files in `$ARCROOT/restart` and resubmit the run.

#### 7.1.3 Ocean model stops due to ocean non-convergence or time-stepping problem

If the ocean log file reports non-convergence and the integration stops, this represents an inability of the ocean model to converge to a solution in the barotropic solver. The best solution is to reduce the ocean model timestep. To do this, edit `scripts/$CASE/ocn.setup.csh`. Increase `DT_COUNT` by

approximately 10% and restart the job. If nonconvergence occurs on the first ocean timestep, then other circumstances may be affecting the ocean-model forcing. Check to see if the proper component forcing files are in place.

It may be necessary to reset the restart files. This is accomplished by copying in the appropriate set of restart and rpointer files into the \$ARCROOT/restart directory. An appropriate set is often available as a tar file in the \$ARCROOT/restart.tars directory.

The coupler will adapt to the change in ocean timestep automatically. Changing the ocean timestep will change the answers and may result in a performance degradation in the coupled model. Normally, the ocean timestep is decreased for a short time and then the timestep is set back to the original value. If the ocean model stops often, it may be worth considering changing the ocean timestep permanently for the model run. Large changes in the ocean timestep may require changes in the model's load balance (\$NTHRDS and \$NTASKS in test.a1.run) for optimal performance.

Occasionally, the ocean model will stop due to a CFL instability. The solution to this problem is the same: reduce the ocean model timestep as described above.

#### 7.1.4 Ice model stops due to ice mpdata transport instability

If the ice log file reports an mpdata transport instability and stops, this indicates an instability in the advection scheme. To solve this problem, increase the value of **ndte** in the scripts/\$CASE/ice.setup.csh file and resubmit the job. Normally, **ndte** is doubled for a short period (a few months or a year) and then reset to the original value. Increasing **ndte** increases the amount of subcycling in the advection scheme. If unstable ice transport is a regular problem, it might be worth increasing **ndte** and leaving it at the larger value for the duration of the model run. The coupler will not need to adapt to an increase in ice subcycling. However, changing the ice subcycling will change the answers and may result in a performance degradation in the coupled model.

If it is necessary to reset the restart files, position the appropriate set of restart and rpointer files into \$ARCROOT/restart directory. An full set of restart and rpointer files is often available as a tar file in the \$ARCROOT/restart.tars directory.

#### 7.1.5 Atmosphere model stops due to ocean non-convergence or time-stepping problem

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 wavenum-
ber 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.

For the standard model, the solution is to increase the dif4 coefficient in atm.setup.csh and run through the instability period (2 months) then change back to the original dif4 coefficient. An alternative is to temporarily increase kmxhdc in atm.setup.csh (default is 5 and maximum value is plev-1) in the same manner. Finally, if the blowup occurs in the first couple of days of an startup run, the CAM namelist option divdampn can be used to easily temporarily increase the divergence dampening (a typical value to try would be 2.).

## 7.2 Debugging

Currently, no debuggers work well with the CCSM model, so print statements are the primary debugging tool. Calls to the shr\_sys\_flush routines (found in \$CSMROOT/models/csm\_share/shr\_sys\_mod.F90) are recommended to ensure that all contents of the print buffers are flushed to standard out before the model halts.

### 7.3 Performance issues

CCSM performance is dependent on a number of issues, including component model configuration, resolution, model timestep, and relative performance of floating-point operations versus memory access speed versus communication latencies on the hardware. These factors affect not only the individual component performance but also the load balancing of the overall configuration.

Each component in CCSM is run as a separate executable and the components communicate with the coupler at regular intervals. These communication points represent intermodel synchronization points. In order to load balance the overall model and allocate appropriate resources for individual components, timings of components between the synchronization points are needed. Most components print out timing information at the end of the run that can be analyzed for load balance and overall performance.

This section will be improved in the future. For help with performance questions, feel free to contact NCAR by e-mailing your questions to [ccsm@ucar.edu](mailto:ccsm@ucar.edu).

## 8 Supporting Scripts:

A number of supporting scripts are included in the CCSM2.0.1 release. Some of these scripts are found in the `scripts/test.a1` directory and others are located in the `scripts/tools` directory. Scripts in the `test.a1` directory are experiment-specific and may need to be changed on an experiment by experiment basis. Scripts in the `tools` serve as a suite of utilities that are intended to be experiment- and machine-independent.

### 8.1 Experiment dependent scripts

The `SCRIPTS/ccsm_archive` and `SCRIPTS/test.a1.har` scripts are supporting scripts to the main CCSM run script `.` Both of these scripts are run from the main script, `SCRIPTS/test.a1.run`. By default, calls to these scripts are currently commented out. These scripts are used mainly for production runs. In addition, there is normally a script call `ccsm_joe` that is created by the main script, `test.a1.run`, to store `job operating environment (joe)` variables and make them available to other supporting scripts.

- `SCRIPTS/ccsm_archive` is executed in the main script, `test.a1.run`, after the CCSM has finished running. The purpose of this script is to move data from the executable directories to a separate directory in order to stage the data for harvesting. In particular, the archiving process moves data from `$EXEROOT` to `$ARCROOT`. `ccsm_archive` gets many of the environment variables from `ccsm_joe`. Then, on a model-by-model and file-by-file basis, `ccsm_archive` copies or moves data from the executable directories to the archive directories. Users may need to change the filenames to be archived for the different models depending on the particular CASE setups.
- `SCRIPTS/test.a1.har` is the ccsM harvester script and it is submitted from the main script as a single-processor, serial, job. It can also be run interactively. `test.a1.har` checks to see whether a file has been harvested by first acquiring the file from the local mass store and then differencing it with the local copy in the archive directory. `test.a1.har` then either removes the local file or copies the local copy of the file to the mass store. It also has the ability to copy files to another location via the Unix copy (`cp` or `scp`) commands.
- `SCRIPTS/ccsm_joe`. which contains many experiment dependent variables for use by other scripts. is generated by the main script, `SCRIPTS/test.a1.run`, every time it's executed. It also serves as a point of reference for checking important environment variables. The `ccsm_joe` script is generated by executing `scripts/tools/ccsm_checkenvs`.

### 8.2 Scripts in the tools directory

Under the `scripts` directory is a `tools` directory. There are several generic utility scripts in this directory. The purpose of these scripts is to provide a generic interface to mass storage systems at different sites, to encapsulate a suite of commands into one utility, to provide a hierarchical search capability for files, and to provide machine-specific commands or notes necessary for running the code at other sites. Scripts in the `tools` directory are meant to be completely case independent. Some of the scripts require case-specific information, which is provided by the `ccsm_joe` script. As mentioned above, these utility scripts are meant to be generic. However, they have been tested on a limited subset of the current available hardware and therefore some limited modifications may be necessary. In particular, the interface to the local mass storage system may need to be altered in these scripts in order for them to work properly for new hardware.

A brief description of the scripts in the `tools` directory is provided below. The scripts are also self-commenting to help the user understand and modify them.

- `ccsm_checkenvs` Echoes environment variables associated with a CCSM run. This script is used to create the `ccsm_joe` file in the main run script. This script has no arguments.

- **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 file. Search for a file in several sources including locally, on the mass store, and on other machines. Copy that file into dir2/file2. Takes one or two arguments.
- **ccsm\_getinput** [**dir1/**]**file1** [**dir2/**]**file2** Hierarchical search for a ccsim input file. Assumes the file will exist under the specific directory "inputdata". dir1 represents the directory path under inputdata to search for the file. Searches locally, in the local inputdata file system, on the local mass store, then at other sites for the file. Takes one or two arguments.
- **ccsm\_getrestart** Copies restart files from a specific restart archive directory into the working directories of the components. This will eventually be configured to look for restart files on the local mass store as well. No arguments.
- **ccsm\_msmkdir** **mssdir** Creates a directory on the local mass store. Takes one argument.
- **ccsm\_msread** [**mssdir/**]**file1** [**loaddir/**]**file2** Copies a file from the local mass store to the local disk. Takes one or two arguments.
- **ccsm\_mswrite** [**loaddir/**]**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.
- **modules.\*.\*** Machine-specific setup scripts to run ccsim. No arguments. Called specifically by the main ccsim script.
- **test.a1.mods.\*** These are notes about how to change the main test script to get it running on other machines.

## 9 The Graphical User Interface: `ccsm_gui`

The CCSM Graphical User Interface (GUI) will generate example scripts to run the CCSM for various time periods in configurations different from the default “all active models” configuration in the CCSM2 distribution. To do this, the GUI modifies the `scripts/test.a1/test.a1.run` example script so that it will run the selected model configurations.

The GUI is a perl wrapper for some of the tools created by the CCSM developers at NCAR for automated generation of different model configurations. While the GUI is designed to generate scripts that run at NCAR, the resulting scripts are a good starting point for off-site users as well.

Currently, the only officially supported configuration and resolution combination is all active components (Configuration B) at the high and low resolution. While the GUI will generate test scripts for other combinations, the results have not been scientifically validated.

The CCSM GUI can be found in the `.../ccsm2.0.1/scripts/gui_run` directory. The GUI is designed so that it does not have to be run on the same machine that the CCSM model will be run. PerlTK is required to run the GUI. Start the GUI by typing `./ccsm_gui` and a window will be created with a few pull-down menus and some input and check boxes. The GUI will search for any previous GUI settings archived in the file `/ccsm_gui`.

First select the platform that the CCSM will be run on: `aix` for IBM SP AIX and `sgi` for SGI Origin 2000-type machines. If the target machine is neither of these, chose `'aix'` if the target machine is a distributed memory architecture and choose `'sgi'` if it is a shared memory architecture. Then select the desired resolution.

Information on the directory structure of the target machine where the CCSM will be run needs to be supplied. Given the proper information, the CCSM gui will generate a new directory containing a CCSM model run script corresponding to the options that were selected.

1. **Select CCSM climate period** specifies the climate period to configure the CCSM for. Standard is present day climate (recommended). Paleoclimate reproduces the T31\_gx3v4 b20.604gm paleoclimate configuration control run (which used 1990 orbital parameters and a fix to the Gent-McWilliams eddy transport parameterization and isopycnal diffusion file (`hmix_gm_fix` in `scripts/test.a2/src.ocn/hmix_gm.F`)).
2. **Select target architecture** specifies the computer platform that the model will be run on. Selecting IBM results in a version compatible with a 4-processor per node IBM-SP. Selecting SGI creates a version for SGI Origin 2000 or 3000 models.
3. **Select CCSM resolution** The CCSM2.0 release officially supported only one resolution (referred to as “T42\_gx1v3”), and one configuration (fully active (B)) with respect to scientific results. This combination was used in the CCSM2.0 Control Run and is the default configuration in the `test.a1` case. The CCSM2.0.1 release supports a second, coarser resolution, “T31\_gx3v4” version of the fully active coupled model. “T31” and “T42” are short-hand references to the resolution of the atmosphere and land models; “gx1v3” and “gx3v4” are short-hand references to the resolution of the ocean and ice models. In addition to these two scientifically validated configurations, a number of other configurations are available for scientific testing and research. More information on the CCSM resolution can be found at <http://www.cesm.ucar.edu/models/ccsm2.0.1/ccsm2.0.1.faq.html>

4. **Select a Configuration** selects the desired combination of active and data model components. The B configuration (all active components) is used for most CCSM runs. The G (active ice, active ocn, data latm and data lnd components) and M (active ice, mixed-layer ocn, data latm and data lnd components) configurations are used to test the sea-ice model and require a T62\_gx1v3 resolution. Any of the other configurations can be built at either T42\_gx1v3 or T31\_gx3v4. Results for configurations other than a B configuration have not yet been scientifically verified.
5. **Select stop time for the run** allows the user to specify the length of the model run. The choices are "stop the run after 5 days", "stop the run after 10 days", "stop the run at a new month", "stop the run at the half year", "stop the run at the new year"
6. **Enter ccsm root directory (CSMROOT)** (string) specifies the top level directory where the ccsm code and scripts reside. The scripts will expect that the complete CCSM2.0.1 code distribution can be found under this directory. For example, if CSMROOT is set to `/home/user/ccsm2.0.1` then the scripts expect that the following directories exist: `/home/user/ccsm2.0.1/models` and `/home/user/ccsm2.0.1/scripts` CSMROOT must be entered by the user.
7. **Enter case name for run (CASE)** (string) specifies the CASE name that identifies this CCSM run. Two scripts will be generated in the directory `CSMROOT/scripts/CASE`: `CSMROOT/scripts/CASE/CASE.run` and `CSMROOT/scripts/CASE/CASE.har`. `CASE.run` corresponds to the one month startup run script. `CASE.har` corresponds to the harvesting script Archiving and harvesting are by default turned off in `CASE.run` (the lines are commented out). To activate these features, the user must edit the script appropriately. If the file `CSMROOT/scripts/CASE/CASE.run` already exists, the gui will not generate the run script and will issue a warning.
8. **Enter ccsm data directory (CSMDATA)** (string) is the root directory for the CCSM input and boundary datasets. This directory differs from the others in that it is not created by the CCSM run script. It is assumed that CSMDATA already exists and contains the data files contained in the CCSM2.0.1 input data distribution tar files. For example, if CSMDATA is set to: `/home/user/input_data` Then the scripts will expect the following input data directory structures to exist underneath `/home/user/input_data`: atm, cpl, ice, lnd, ocn. CSMDATA is where the input data tar files are untarred to.
9. **Enter ccsm executable directory (EXEROOT)** (string) is the directory where the model will run. Subdirectories for each of the model components will be created under EXEROOT. Typically, this is a local, fast, and large temporary disk space. For example, if EXEROOT is set to `/ptmp/user/CASE` Then, the following directories will be generated by the script at run time below EXEROOT:all, atm, cpl, esmf, ice, lib, lnd, ocn. The model will be run in these directories.

## 10 The Atmosphere Setup Script: atm.setup.csh

The atmosphere setup script, `SCRIPTS/atm.setup.csh`, is called by `SCRIPTS/test.a1.run` when the atm model is selected in the `MODELS` environment variable array. The default atmosphere model is NCAR's Community Atmosphere Model (CAM). The CAM is the successor to the earlier Community Climate Model (CCM). The `atm.setup.csh` script prepares the atmosphere model component for execution. To do so, it defines the run environment, positions the atmosphere input data sets, then builds the dynamic atmosphere model by calling `gmake`.

If `SCRIPTS/atm.setup.csh` is unable to complete any of these tasks, it will abort with a non-zero error status. The `test.a1.run` script checks the error status and will halt if an error is detected.

### 10.1 Document the atmosphere setup script

```
#!/bin/csh -f
#=====
# CVS $Id: atm.setup.csh.tex,v 1.9 2002/10/10 19:17:00 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/atm.setup.csh.tex,v $
# CVS $Name: $
echo =====
echo atm.setup.csh: Preparing CAM for execution 'date'
echo Purpose:
echo a. gather or create necessary input files
echo b. build the CAM, a CCSM active atm component
echo documentation: www.cesm.ucar.edu/models/ccsm2.0.1
echo =====

cat $0;$TOOLS/ccsm_checkenvs || exit -1 # check that enviroment variables are set
```

This first section documents the atmosphere setup script.

The first line of this section identifies this as a C-shell script. The `-f` option prevents the user's personalized `$HOME/.cshrc` file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The cat command combines two functions on one line. The `cat $0` command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then `$TOOLS/ccsm_checkenvs` writes the environment variables that have been set by `test.a1.run` into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.



## 10.2 Define and position the atmosphere input datasets

```
echo '-----'
echo ' (a) gather or create necessary input files in EXEDIR      '
echo '-----'

cd $EXEDIR

# Set restart type
#-----
if ($RUNTYPE == 'startup' ) set NSREST = 0
if ($RUNTYPE == 'continue') set NSREST = 1
if ($RUNTYPE == 'branch' ) set NSREST = 3
if ($RUNTYPE == 'hybrid' ) set NSREST = 0

# specify physics version: physics
#-----
set PHYSICS = physics
```

First, the `cd $EXEDIR` command makes \$EXEDIR the current working directory. The `test.a1.run` script defines \$EXEDIR as \$EXEROOT/atm . All of the atmosphere input and output files will be read from and written to the \$EXEDIR directory. Then the runtime setting is parsed into NSREST values appropriate for the atmosphere model. Finally, the physics version is specified.

```

# input datasets
#-----
set DATOZON = noaa03.1990.21999.nc
set ABSDATA = abs_ems_factors_fastvx.052001.nc
$TOOLS/ccsm_getinput atm/cam2/ozone/$DATOZON || exit 99
$TOOLS/ccsm_getinput atm/cam2/rad/$ABSDATA || exit 99

# Set parameters associated with resolution and find input dataset
#-----

set DYNAMICS = eul
set NREVSN = ""
set DATINIT = ""

if ($RUNTYPE == 'startup') then
  if ($GRID =~ T85*) set DATINIT = JAN1.T85L26.eul.cesm.c011015.nc
  if ($GRID =~ T42*) set DATINIT = JAN1.T42L26.eul.cesm.052001.nc
  if ($GRID =~ T31*) set DATINIT = JAN1.T31L26.eul.cesm.c011018.nc
  if ($GRID =~ T21*) set DATINIT = SEP1.T21L26.c011017.nc
  $TOOLS/ccsm_getinput atm/cam2/inic/gaus/$DATINIT || exit 99
  if !($?DATINIT) then
    echo atm: ERROR unsupported resolution/physics/dynamics combination
    exit 2
  endif
endif
if ($RUNTYPE == 'branch') then
  set NREVSN = ${REFCASE}.cam2.r.${REFDATE}-00000
  $TOOLS/ccsm_getfile $REFCASE/atm/rest/$NREVSN || exit 99
endif
if ($RUNTYPE == 'hybrid') then
  set DATINIT = ${REFCASE}.cam2.i.${REFDATE}-00000.nc
  $TOOLS/ccsm_getfile $REFCASE/atm/init/$DATINIT || exit 99
endif

```

This section positions the input datasets needed for the atmosphere model. The input datasets are all uniquely named by a description and a six digit number which documents the creation date (format: yymmdd) of the file. This date-stamp in the filename ensures that the exact data file requested by the user is input into the model.

Tools from the \$TOOLS directory make their first appearance here. \$TOOLS/ccsm\_cpdata will attempt to copy a version of the file from a local disk into the current working directory. If a local copy of the data file is unavailable, the script will abort. In the branch conditional, \$TOOLS/ccsm\_msread is used to bring in the restart data set from the sites mass storage device. The complete range of CCSM tools scripts are described in the **Supporting Scripts** section of this document.

DATOZON is a 4-dimensional(time,longitude, latitude,level) time-series ozone dataset. ABSDATA is the absorptivity and emissivity dataset. DATINIT contains the initial atmospheric condition of the model. Alternative versions for different resolutions are also defined.

If this is a branch run, the year, month and day are extracted from the reference date (\$REFDATE) and used to acquire the necessary atmosphere restart file.

### 10.3 Define resolution dependent parameters

```

if ($GRID =~ T85*) set PARAMS = (85 256 128 26 .true. .false. 600 1.5e15)
if ($GRID =~ T42*) set PARAMS = (42 128 64 26 .true. .true. 1200 1.0e16)
if ($GRID =~ T31*) set PARAMS = (31 96 48 26 .true. .false. 1800 2.0e16)
if ($GRID =~ T21*) set PARAMS = (21 64 32 18 .true. .false. 2400 2.0e16)

set plon = $PARAMS[2]; set plat = $PARAMS[3] ; set plev = $PARAMS[4]
set ptrm = $PARAMS[1]; set ptrn = $PARAMS[1] ; set ptrk = $PARAMS[1]
set FLXAVE = $PARAMS[5]; set READTRACE = $PARAMS[6]; set DTIME = $PARAMS[7]
set DIF4 = $PARAMS[8];

set KMXHDC = 1
if ($GRID =~ T31*) set KMXHDC = 5
if ($GRID =~ T42* && $plev != 18) set KMXHDC = 5
if ($GRID =~ T85* && $plev != 18) set KMXHDC = 5

if ($DYNAMICS != 'eul') then
  set DIF4 = 0.
endif

```

Here many of the resolution dependent settings of the atmosphere model are defined. For each resolution, the PARAMS array contains the settings for the spectral truncation, the longitude, latitude and level dimensions, and settings for flux averaging, input tracers and the model time step.

In addition, KMXHDC, which defines the number of vertical levels to which the Courant limiter is applied (Defaults to 1.), is set. These settings are used both to help build the input namelist and in the building of the model executable.

## 10.4 Write the atmosphere input namelist

```
# Create an input namelist file
#-----
set BASEDATE_NUM = `echo $BASEDATE | sed -e 's/-//g'` # remove "-"

cat >! atm.stdin << EOF
&camexp
caseid      = '$CASE'
ctitle      = '$CASE $CASESTR'
ncdata      = '$DATINIT'
bndtvo      = '$DATOZON'
nrevsn      = '/$MSSNAME/csm/$REFCASE/atm/rest/$NREVSN'
absems_data = '$ABSDATA'
mss_irt     = $MSSRPD
mss_wpass   = '$MSSPWD'
nsrest      = $NSREST
dtime       = $DTIME
dif4        = $DIF4
start_ymd   = $BASEDATE_NUM
linebuf     = .false.
readtrace   = $READTRACE
nelapse     = -9999
ozncyc      = .true.
flxave      = $FLXAVE
kmxhdc      = $KMXHDC
rest_pfile  = '$SCRIPTS/rpointer.$MODEL'
/
EOF
```

This section constructs the input namelist which is used to control CAM runtime operation. First, the start date, then the runtime setting is parsed into variables which the atmosphere model can interpret.

The primary interface to the atmosphere model is through the namelist input file, in which a wide range of predefined parameters are set to control the behavior of the atmosphere model. The namelist input file, called atm.stdin, is a text file which is read by the atmosphere model. Namelist input for the atmosphere consists of text strings enclosed in quotes, integer and real numerical values and logicals.

A few of the CCSM environment variables, such as \$BASEDATE and \$RUNTYPE, are translated into variables recognized by the CAM.

The "cat" command uses the C-shell here-document option to create the file \$EXEDIR/atm.stdin. When atm.stdin is created, all of the environment variables are set to the current values as defined by the \$SCRIPTS/test.a1.run script.

<b>&amp;camexp</b>	is the namelist group name, which matches the groupname defined within CAM.
<b>caseid</b>	= <b>\$CASE</b> (string) sets a text string (16 characters or less) that is used to identify this run. The CASE variable is set in \$SCRIPTS/test.a1.run and is used extensively in the CCSM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX file-naming characters such as letters, numbers, underscores, dashes, commas or periods.
<b>ctitle</b>	= <b>'\$CASE \$CASESTR'</b> (string) provides 80 characters to further describe this run. This description appears in the output logs and in the header data for the output data sets. CASESTR is set in the \$SCRIPTS/test.a1.run script.
<b>ncdata</b>	= <b>'\$DATINIT'</b> is the name of the dataset which contains the initial-condition data for the atmosphere. It contains three-dimensional fields of T, U, V, Q and two-dimensional fields of surface geopotential, standard deviation of orography, land ocean transition flags and masks, subsoil temperature profiles, and constituent tracers.
<b>bndtvo</b>	= <b>'\$DATOZON'</b> (string) is the name of the ozone boundary dataset.
<b>nrevsn</b>	= <b>'\$NREVSN'</b> (string) is the optional name of the restart dataset to be read if this is a branch run. This dataset is a binary file containing the exact state of a previous run.
<b>absems_data</b>	= <b>'\$ABSDATA'</b> (string) is the name of the absorptivity and emissivity dataset.
<b>irt</b>	= <b>\$MSSRPD</b> and <b>rirt = \$MSSRPD</b> (integer) specifies the mass store retention period for output history and restart files respectively when running at NCAR. If irt or rirt are set to 0, the output history or restart files are not written to the NCAR MSS.
<b>nswrps</b>	= <b>'\$MSSPWD'</b> (string) sets the write password for any files written to the NCAR MSS.
<b>nsrest</b>	= <b>\$NSREST</b> (string) specifies the state in which the run is being started. nsrest settings 0,1,3 map into the CCSM variables (startup/hybrid, continue and branch respectively).
<b>dttime</b>	= <b>\$DTIME</b> (integer) sets the time step, in seconds, for the atmosphere model. The time step is dependent on both the resolution of the model and the dynamical core used by the model.
<b>start_ymd</b>	= <b>\$START_YMD</b> (integer) specifies the "basedate" for the run. This date serves as the baseline from which the current step number is calculated.
<b>linebuf</b>	= <b>.false.</b> (logical) turns off line-by-line buffering in favor of the default buffering. The linebuf option allows the user to control whether standard output is written line-by-line or buffered. Setting linebuf = .false. allows printed output to be buffered, resulting in increased performance. linebuf is typically set to .true. for debugging so that printed output will be written to the standard out device immediately, rather than being buffered.
<b>readtrace</b>	= <b>\$READTRACE</b> (logical) specifies whether or not an input chemical tracer file is read.
<b>nelapse</b>	= <b>-9999</b> (integer) indicates that the flux coupler will control the ending timestep for this model run.
<b>ozncyc</b>	= <b>.true.</b> (logical) will result in the input ozone dataset being cycled through the first 12 months of data in the input ozone file.

**flxave** = **\$FLXAVE** (logical) controls whether or not the atmosphere flux averaging feature is enabled.

**kmxhdc** = **\$KMXHDC** (integer) specifies the number of levels over which to apply Courant limiter, starting at top of model.

**rest\_pfile** = **'\$SCRIPTS/rpointer.\$MODEL'** (string) is the filename of the restart pointer file. The restart pointer file is a one line text file containing the name of the CAM restart files needed to continue the run.  
/ marks the end of the camexp namelist input variables.

**EOF** marks the end of the unix here document referenced by the cat command near the start of this section.

Detailed information on the complete range of atmospheric namelist variables can be found in the CAM User's Guide (<http://www.cgd.ucar.edu/cms/atm-cam>).

## 10.5 Create the atmosphere executable

The remainder of the atm.setup.csh builds the CAM executable. First the location of the source code is specified, then various resolution and configuration information is put into header files. Once the source code Filepath and header files have been created, the CAM executable is built with gmake.

```

echo '-----'
echo ' (b) Build an executable '
echo '-----'

cd $OBJDIR

# atm source code archive
#-----
set SRCDIR = $CSMCODE/atm/cam/src
set ATMLNDSHR = $CSMCODE/camclm_share
set MATHUTIL = mathutil

```

To avoid filling the \$EXEDIR directory with all the artifacts of the build process, the model is built in a subdirectory, \$OBJDIR. \$OBJDIR which is defined in the \$SCRIPTS/test.a1.run script.

\$SRCDIR identifies the root directory of the atmospheric source code and \$ATMLNDSHR points to the directory containing CSM code shared by both the CAM and Common Land Model (CLM) components.

The variable CSMCODE is set in the test.a1.run script to coordinate CCSM source code.

```
# Filepath: List of source code directories (in order of importance).
# The cmp command will force a rebuild the old and new files are different
#-----
\cat >! .tmp << EOF; cmp -s .tmp Filepath || mv -f .tmp Filepath
$SCRIPTS/src.atm
$SRCDIR/dynamics/$DYNAMICS
$SRCDIR/advection/slt
$SRCDIR/control
$SRCDIR/$MATHUTIL
$SRCDIR/$PHYSICS
$ATMLNDSHR
$CSMSHR
$CSMCODE/utils/timing
EOF
```

The Filepath file contains the list of source code directories from which to gather the input source code. This list will be used as the input to the gmake VPATH list.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.atm, is typically used to hold modified atmospheric source code. If a directory in the Filepath list is either empty or doesn't exist at all, no error will result.

To avoid a complete rebuild of the model every time this script is called, the Filepath list is first put into a temporary file, then the contents of the temporary file are compared against any existing Filepath list. If there is no existing Filepath list or the temporary file differs from the existing Filepath list, the new tmp file is renamed Filepath, causing a complete rebuild of the atmosphere model. If the temporary file is exactly the same as the existing Filepath file, nothing happens.

```

# build params.h      (update only if new or changed)
#-----

\cat >! .tmp << EOF; cmp -s .tmp params.h || mv -f .tmp params.h
#ifndef PARAMS_SET
#define PARAMS_SET
#define PCNST      1
#define PNATS      1
#define PLEV       $plev
#define PLEVR      $plev
#define PLON       $plon
#define PCOLS      16
#define PLAT       $plat
#define PTRM       $ptrm
#define PTRN       $ptrn
#define PTRK       $ptrk
#endif
EOF

# build misc.h        (update only if new or changed)
#-----

if ($NTASK == 1) set SPMD = "#undef  SPMD"
if ($NTASK > 1) set SPMD = "#define SPMD"

\cat >! .tmp << EOF; cmp -s .tmp misc.h || mv -f .tmp misc.h
#ifndef MISC_SET
#define MISC_SET
#define COUP_CSM
$SPMD
#endif
EOF

```

Two CAM include files are created here. `params.h` contains information about the model dimensions while `misc.h` contains flags for whether the model is coupled or standalone and whether CAM will be built to run multitasked.

```

# Create the executable in EXEDIR/obj
#-----

#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c
  gmake -j 4 -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS MODEL=cam \
    VPPATH=Filepath EXEC=$EXEDIR/atm || exit 2
else
  echo "BLDTYPE = $BLDTYPE"
endif

```



In this last section the atmosphere model is built using gmake.

The CCSM uses the gnumake (also known as “gmake”) tool to build the model executable. Each of the components setup scripts creates a list of source code directories from which to gather the input source code for that component. This list is called Filepath and will be used as the input to the gmake VPATH list. The file Filepath is written in each of the components \$OBJDIR directories.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.atm, is typically used to hold modified source code. If a directory in the Filepath list is either empty or doesn't exist at all, no error will result. In general, the directories \$SCRIPTS/src.\$MODEL can be used to store locally modified source code. Each component script recognizes this directory as the top priority for finding source code.

First the makdep code is compiled. This utility program is called by the Makefile and checks for source code dependencies. This is done by seeing if any of the header or include files have been updated since the model was last built and ensures that the F90 modules are constructed in the proper order.

Once makdep is compiled, the GNU make program, gmake, is used to actually build the model. The -j 4 option uses 4 processors to build the model. The -f \$CSMBUILD/Makefile points to the generic CCSM Makefile while MACFILE=\$CSMBLD/Macros.\$OS points to the machine specific make options. MODEL identifies the component being built and VPATH points to the Filepath list. Finally, the actual executable to be built is \$EXEDIR/\$MODEL.

Once the atm.setup.csh script completes successfully, it will exit with a 0 (no-error) status.

## 11 The Ocean Model Setup Script: ocn.setup.csh

The ocean setup script, \$SCRIPTS/ocn.setup.csh, is called by test.a1.run when an active ocean model is selected in the \$MODELS environment variable array. ocn.setup.csh prepares the ocean model component for execution. To do so, it defines the run environment, collects the ocean input data sets, then builds the dynamic ocean model by calling gmake.

If ocn.setup.csh is unable to complete any of these tasks, it will abort with a non-zero error status. The test.a1.run script checks the error status and will halt if an error is detected.

### 11.1 Document the ocean setup script

```

#!/bin/csh -fv
#=====
# Purpose:
# (a) build an executable model (pop, a CSM active ocn component)
# (b) gather or create necessary input files
#
# For help, see: www.cesm.ucar.edu/models/ccsm2.0.1
#=====
cat $0;$TOOLS/ccsm_checkenvs || exit -1          # cat this file, check envs

```

This first section documents the ocean setup script.

The first line of this section identifies this as a C-shell script. The "-f" option prevents the user's personalized \$HOME/.cshrc file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The cat command combines two functions on one line. The "cat \$0" command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then \$TOOLS/ccsm\_checkenvs writes the environment variables that have been set by test.a1.run into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.

## 11.2 Set code location and resolution dependencies

```

echo -----
echo  setting up some internal Env variables and path names
echo -----

set SRCDIR  = $CSMCODE/ocn/pop
set my_path = $SCRIPTS/src.ocn

if ( $OCN_GRID ==  gx1v3) then
  setenv DT_COUNT  23
  setenv TIME_MIX_FREQ  17
  setenv INIT_TS_FILE ts_PHC2_jan_20010711

else if ( $OCN_GRID ==  gx3v4) then
  setenv DT_COUNT  12
  setenv TIME_MIX_FREQ  17
  setenv INIT_TS_FILE ts_PHC2_jan_20020617

else
  echo "$0 ERROR: Cannot deal with GRID = $GRID"
  exit -1
endif

```

This section defines the ocean model source-code location and sets a few resolution dependent variables.

**SRCDIR** points to the directory for the POP model. Under **SRCDIR** are a number of subdirectories containing source code, documentation, input templates and other files needed by POP. **SRCDIR** is typically treated as an archive directory containing “frozen” code. Frozen source code refers to code that represents the exact released version of the model. Frozen code is never modified under any circumstances. To modify code, copy it from **SRCDIR** into **\$my\_path** and make the necessary changes to the **\$my\_path** copy.

**\$my\_path** identifies a directory that contains user modified POP code. Any POP code that a user needs to modify should be placed in the **\$my\_path** directory. POP source code files in **\$my\_path** will have precedence over any copies of the same file that exist under **SRCDIR**. **\$my\_path** is designed to be only one level deep, with no subdirectories. If changes are so extensive that subdirectories are required, the ocn.setup.csh script will need to be modified accordingly.

Next, three resolution dependent variables are set. These variables will be input into the POP namelist via the pop\_in file, which is modified in section b:

**DT\_COUNT** is the number of timesteps per day.

**TIME\_MIX\_FREQ** is the number of timesteps between time-mixing

**INIT\_TS\_FILE** is the initial-condition temperature and salinity restart file used for continuation or startup runs.

See the POP User’s Guide (<http://www.cesm.ucar.edu/models/ccsm2.0.1/pop/doc> ). for full details.

### 11.3 Position initial files

```

# branch- or hybrid-run setup
#-----
if ($RUNTYPE == branch || $RUNTYPE == hybrid) then
  setenv INIT_TS_FILE ${REFCASE}.pop.r.$REFDATE-00000
  setenv TAVG_TS_FILE ${REFCASE}.pop.rh.$REFDATE-00000

# position the branch or hybrid files
if !(-d rest ) mkdir -p rest
cd rest
$TOOLS/ccsm_getfile $REFCASE/ocn/rest/$INIT_TS_FILE      || exit 99
$TOOLS/ccsm_getfile $REFCASE/ocn/rest/$INIT_TS_FILE.hdr  || exit 99
$TOOLS/ccsm_getfile $REFCASE/ocn/rest/$TAVG_TS_FILE      || exit 99
$TOOLS/ccsm_getfile $REFCASE/ocn/rest/$TAVG_TS_FILE.hdr  || exit 99
cd ..

# generate the appropriate restart pointer files
cat >! $SCRIPTS/rpointer.ocn.restart << EOF
$cwd/rest/$INIT_TS_FILE.hdr
$cwd/rest/$INIT_TS_FILE
EOF

cat >! $SCRIPTS/rpointer.ocn.tavg << EOF
$cwd/rest/$TAVG_TS_FILE.hdr
$cwd/rest/$TAVG_TS_FILE
EOF

endif

```

In this section, a number of the initial data and restart pointer files are identified and positioned for branch or hybrid runs. For hybrid or branch runs, the initial (INIT\_TS\_FILE) and time-averaged (TAVG\_TS\_FILES) input filenames are determined by resolving the name based on the \$REFCASE and \$REFDATE variables set in the test.a1.run script. These data files and their associated header files are placed in the ocean restart directory. Finally the restart pointer files, \$SCRIPTS/rpointer.ocn.restart and \$SCRIPTS/rpointer.ocn.tavg, are created.

## 11.4 Build the executable

```

echo -----
echo a. Build an executable in $OBJDIR
echo -----

if ( -e ocn && $?DONTBUILD ) then
  echo "Note: using an existing binary"
else

  echo -----
  echo a.1 Calculate processor tiling based on $NTASK
  echo -----
  if ($NTASK == 4)  setenv NX 2
  if ($NTASK == 8)  setenv NX 4
  if ($NTASK == 16) setenv NX 4
  if ($NTASK == 32) setenv NX 8
  if ($NTASK == 40) setenv NX 5
  if ($NTASK == 48) setenv NX 8
  if ($NTASK == 64) setenv NX 8
  if !($?NX)        echo ERROR: ocn cant deal with NTASK=$NTASK
  if !($?NX)        exit -1
  @ ny = $NTASK / $NX; setenv NY $ny

```

The POP executable is built in this section. First the processor tiling is resolved, then the code necessary to build the model is copied in from a number of locations. Once the source code is in place, the POP executable is built with gmake.

This section attempts to automatically specify, for a variety of pre-determined processor counts, the values of NX and NY, the number of processors to assigned to the X and Y dimensions for a number of pre-determined processor counts. This is designed to be automatic. The values of NX and NY are passed as arguments to the gmake file, where they are used to set the values of the C-preprocessor parameters NPROC\_X and NPROC\_Y. Further documentation on NPROC\_X and NPROC\_Y is in the POP documentation: [http://www.cesm.ucar.edu/models/ccsm2.0.1/pop/doc/POPusers\\_chap2.html](http://www.cesm.ucar.edu/models/ccsm2.0.1/pop/doc/POPusers_chap2.html)

```

echo -----
echo a2. creating the internal directory structure
echo -----

set compile_dir = $OBJDIR/compile
set source_dir  = $OBJDIR/source
if !(-d $source_dir ) mkdir -p $source_dir
if !(-d $compile_dir ) mkdir -p $compile_dir

echo ' '
echo '-----'
echo ' copy the necessary files into $source_dir '
echo '-----'
cd $source_dir
cp -fp $SRCDIR/input_templates/${OCN_GRID}_model_size.F model_size.F
cp -fp $SRCDIR/source/*. [FCc] .
cp -fp $SRCDIR/mpi/*. [FCc] .
cp -fp $CSMSHR/*. [F]* .
cp -fp $CSMSHR/*. [Cc] .
if (-d $my_path ) cp -fp $my_path/*. [FCc] .

```

The POP source code and compilation products are placed into two different directories, `source_dir` and `compile_dir`. Here, the two directories are created and all the necessary source code is copied into **source\_dir**. Notice that the last copy is from **\$my\_path**, which overwrites any files that have the same name as the files in **\$my\_path**.

`input_templates/gx1v3_model_size.F` specifies the grid dimensions and number of tracers.

```

#
# recompile if 2d decomp is changed
#
echo $OCN_GRID $NTASK $NX $NY >! $OBJDIR/ocnres.new
diff $OBJDIR/ocnres.new $OBJDIR/ocnres.old || touch `grep -l NPROC $OBJDIR/source/*`
echo $OCN_GRID $NTASK $NX $NY >! $OBJDIR/ocnres.old

```

The POP processor tiling is compiled into the model executable. Any changes to the number of processors assigned to POP will change the tiling, which requires recompiling POP. This section compares the currently requested tiling with the tiling specified during the last build. If the two tilings are different, a recompilation of POP is forced.

```

echo -----
echo  a3. compile and copy the executable into $EXEDIR directory
echo  -----

#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1
if ($BLDTYPE == 'true') then
  cd $compile_dir

  cc -o makdep $CSMBLD/makdep.c || exit 2

  set EXEC = ocn_${NTASK}
  set THREAD = FALSE ; if ($NTHRD > 1) set THREAD = TRUE
  gmake -j 6 -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS \
MODEL=pop NX=$NX NY=$NY THREAD=$THREAD \
  VPATH=$source_dir EXEC=$OBJDIR/$EXEC || exit 2

# gmake -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS mostlyclean

#--- cp & link into EXEDIR ---
  rm -f $EXEDIR/ocn
  cp $OBJDIR/$EXEC $EXEDIR/ocn
else
  echo "BLDTYPE = $BLDTYPE"
endif
endif

```

A number of steps are required to build the POP executable using gmake. First, the dependency generator, makdep, is created. When gmake is called, makdep will build a list of source-code dependencies that identify the specific POP files that need to be recompiled if any of the source code is modified.

**\$EXEC** specifies the name of the POP executable. The processor count **NTASK** is appended to the name to differentiate between executables built for different values of **NTASK**.

**\$THREAD** allows for multiple MPI threads. However, since POP does not support threading, this should always be **FALSE**.

Finally, gmake is called using the generic CCSM gmake and macros files and the resulting executable is copied into the ocean-model execution directory.

## 11.5 Parse the date variables

```

echo ' '
echo '-----'
echo " b. gather or create necessary input files in $EXEDIR "
echo '-----'

set IYEAR0      = 'echo $BASEDATE | cut -c1-4 | sed -e 's/^0*//''
set IMONTH0     = 'echo $BASEDATE | cut -c6-7 | sed -e 's/^0*//''
set IDAY0       = 'echo $BASEDATE | cut -c9-10 | sed -e 's/^0*//''

if ($RUNTYPE == startup || $RUNTYPE == hybrid ) @ IDAY0 = $IDAY0 + 1

```

The POP model receives the initial date information in a slightly different format than it is defined in the \$SCRIPTS/test.a1.run script. This short section does this text transformation.

**IYEAR0**, **IMONTH0** and **IDAY0** are the numeric representations for the coordinated base year, month and day set in test.a1.run. Startup and hybrid runs with POP require that the rest of the coupled system run for 1 day before POP starts. This is accomplished by incrementing the IDAY0 setting.

## 11.6 Modify pop\_in

```

# sed commands are of the form s#OldString#NewString#

cat >! $EXEDIR/commands.sed << EOF
s#OUTPUTL#$EXEDIR#
s#OUTPUTR#$EXEDIR/rest/$CASE.pop.r#
s#OUTPUTH#$EXEDIR/hist/$CASE.pop.h#
s#OUTPUTD#$EXEDIR/hist/$CASE.pop.d#
s#OUTPUT/pop_pointer#$SCRIPTS/rpointer.ocn#

s#RUNID#$CASE#;          s#LOG_FILENAME#$ocn_out#
s#INIT_TS_OPTION#$RUNTYPE#
s#INPUT#$EXEDIR/input#
s#DT_COUNT#$DT_COUNT#;  s#TIME_MIX_FREQ#$TIME_MIX_FREQ#

s#IYEAR0#$IYEAR0#;      s#IHOURO#0#
s#IMONTH0#$IMONTH0#;    s#IMINUTE0#0#
s#IDAY0#$IDAY0#;        s#ISECONDO#0#

EOF

sed -f $EXEDIR/commands.sed \
    $SRCDIR/input_templates/${OCN_GRID}_pop_in >! $EXEDIR/pop_in

# put pop_in where pop can find it (specific to mpirun, poe, etc)
ln -sf $EXEDIR/pop_in $EXEDIR/..
if (-d $EXEDIR/./all) ln -sf $EXEDIR/pop_in $EXEDIR/./all

```



Building the input namelist file for the ocean setup procedure differs from the other components. The text file “pop\_in” contains a complete, resolution-dependent set of namelist input to POP, with several generically named values that are intended to be automatically modified by the ocean setup script. This section carries out this modification using the Unix stream editor, sed, to insert the desired namelist values into the pop\_in file.

This set of sed commands modifies selected variables from the large generic pop\_in namelist file in the directory \$SRCDIR/input\_templates. The sed commands are placed in the file commands.sed, then applied against the generic pop\_in file. Additional changes to the pop\_in file can be added here by following the examples in the script and appropriately modifying the pop\_in file. The resulting pop\_in file is linked into the directory in the next level up and to the /all directory for use on the SGI.

## 11.7 Define the ocean input datasets

```

echo '-----'
echo  c. cp input datasets from $SRCDIR ...
echo '-----'

if !( -d $EXEDIR/rest )  mkdir -p $EXEDIR/rest  || exit 2
if !( -d $EXEDIR/hist )  mkdir -p $EXEDIR/hist  || exit 2
if !( -d $EXEDIR/input ) mkdir -p $EXEDIR/input || exit 2

cd $EXEDIR/input

foreach FILE (depth_accel history_contents \
              movie_contents region_ids  scalar_contents tavg_contents \
              transport_contents vert_grid )
  $TOOLS/ccsm_cpdata $SRCDIR/input_templates/${OCN_GRID}_${FILE} $FILE || exit 99
end
foreach FILE (eos_coefficients.ieeer8 )
  $TOOLS/ccsm_cpdata $SRCDIR/input_templates/${OCN_GRID}_${FILE} $FILE
end

```

This section collects the input datasets needed for the ocean model. The input datasets are all uniquely named by a description and a six digit number that documents the creation date (format: yymmdd) of the file. While the hard-wiring of filenames restricts the degree of automation, it ensures that the exact data that the user requests is input into the model.

First, the restart, history and input data directories are created if they don’t already exist. Then text input files from the POP “input\_template” directory are copied into the ocean input directory.

See the CCSM2.0.1 POP User’s Guide, Chapter 4, Model diagnostics and output, for a complete discussion of these files ( [http://www.cesm.ucar.edu/models/ccsm2.0.1/pop/doc/POPusers\\_chap4.html](http://www.cesm.ucar.edu/models/ccsm2.0.1/pop/doc/POPusers_chap4.html) ).

- **input\_templates/gx1v3\_depth\_accel** Contains a list of depth-acceleration factors, one entry per model level (set uniformly to 1, if unaccelerated).
- **input\_templates/gx1v3\_history\_contents** Contains a list of instantaneous history fields to be written to the “snapshot” output file.
- **input\_templates/gx1v3\_tavg\_contents** Contains a list of time-averaged history fields to be written to the “history” output file.

- **input\_templates/gx1v3\_movie\_contents** Contains a list of 2-D slices to be written to a “snapshot” output file. The resulting time series of slices is often used in the creation of movies.
- **input\_templates/gx1v3\_region\_ids** Contains names of ocean regions and marginal sea balancing coordinates.
- **input\_templates/gx1v3\_scalar\_contents** Sets attributes of scalars such as units, short\_name, and long\_name. The short\_name agrees with the associated variable name in the code.
- **input\_templates/gx1v3\_transport\_contents** Specifies section lines across which transports will be computed.
- **input\_templates/gx1v3\_pop\_in** Contains all namelists which will be read by the model.
- **input\_templates/gx1v3\_vert\_grid** Specifies the vertical grid.

## 11.8 Position the ocean input datasets

```

echo '-----'
echo  c. acquire initial/boundary datasets disk or MSS
echo '-----'

set OCNDATA = ocn/pop/$OCN_GRID # point to disk copies

if ($RUNTYPE == startup ) then
$TOOLS/ccsm_getinput $OCNDATA/ic/${INIT_TS_FILE}.ieeer8      ts
$TOOLS/ccsm_getinput $OCNDATA/ic/${INIT_TS_FILE}.readme      ts.readme
endif

if ($OCN_GRID == gx1v3) then
$TOOLS/ccsm_getinput $OCNDATA/grid/horiz_grid_20010402.ieeer8  horiz_grid
$TOOLS/ccsm_getinput $OCNDATA/grid/horiz_grid_20010402.readme  horiz_grid.readme
$TOOLS/ccsm_getinput $OCNDATA/grid/region_mask_20010709.ieeei4  region_mask
$TOOLS/ccsm_getinput $OCNDATA/grid/region_mask_20010709.readme  region_mask.readme
$TOOLS/ccsm_getinput $OCNDATA/grid/topography_20010702.ieeei4  topography
$TOOLS/ccsm_getinput $OCNDATA/grid/topography_20010702.readme  topography.readme
$TOOLS/ccsm_getinput $OCNDATA/forcing/shf_mm_all_85-88_20010308.ieeer8  shf
$TOOLS/ccsm_getinput $OCNDATA/forcing/shf_mm_all_85-88_20010308.readme  shf.readme
$TOOLS/ccsm_getinput $OCNDATA/forcing/sfwf_mm_all_85-88_20010320.readme  sfwf.readme
$TOOLS/ccsm_getinput $OCNDATA/forcing/sfwf_mm_all_85-88_20010320.ieeer8  sfwf

else if ($OCN_GRID == gx3v4) then
$TOOLS/ccsm_getinput $OCNDATA/grid/horiz_grid_20001030.ieeer8  horiz_grid
$TOOLS/ccsm_getinput $OCNDATA/grid/horiz_grid_20001030.readme  horiz_grid.readme
$TOOLS/ccsm_getinput $OCNDATA/grid/region_mask_20020617.ieeei4  region_mask
$TOOLS/ccsm_getinput $OCNDATA/grid/region_mask_20020617.readme  region_mask.readme
$TOOLS/ccsm_getinput $OCNDATA/grid/topography_20020617.ieeei4  topography
$TOOLS/ccsm_getinput $OCNDATA/grid/topography_20020617.readme  topography.readme
$TOOLS/ccsm_getinput $OCNDATA/forcing/shf_20011030.ieeer8      shf
$TOOLS/ccsm_getinput $OCNDATA/forcing/shf_20011030.readme      shf.readme
$TOOLS/ccsm_getinput $OCNDATA/forcing/sfwf_20011030.ieeer8      sfwf
$TOOLS/ccsm_getinput $OCNDATA/forcing/sfwf_20011030.readme      sfwf.readme

endif

wait

```

The variable OCNDATA defines the location of the collection of initial and boundary datasets that are distributed with CCSM2. These variables are:

<b>horiz_grid_20010402.ieeer8</b>	Specifies the horizontal grid.
<b>horiz_grid_20010402.readme</b>	
<b>region_mask_20010709.ieeei4</b>	Specifies the region mask.
<b>region_mask_20010709.readme</b>	
<b>topography_20010702.ieeei4</b>	Specifies the topography (index of deepest ocean level at each horizontal grid point).
<b>topography_20010702.readme</b>	
<b>shf_mm_all_85-88_20010308.ieeer8</b>	Contains monthly averaged fields of sea-surface temperature needed by the surface heat flux forcing option “restoring”.
<b>shf_mm_all_85-88_20010308.readme</b>	
<b>sfwf_mm_all_85-88_20010320.ieeer8</b>	Contains monthly averaged fields of sea-surface salinity needed by the surface freshwater flux forcing option “restoring”.
<b>sfwf_mm_all_85-88_20010320.readme</b>	

The POP input datasets are copied of from \$OCNDATA to the \$EXEDIR/input directory. Note that in the case of a “startup” run, initial temperature and salinity initial conditions are also copied to the input directory.

Once the ocn.setup.csh script completes successfully, it will exit with a status value of 0.

## 12 The Sea-Ice Model Setup Script: ice.setup.csh

The sea-ice setup script, `scripts/test.a1/ice.setup.csh`, is called by `test.a1.run` when the ice model is selected in the `MODELS` environment variable array. The sea-ice model is the CCSM Sea Ice Model version v4 (CSIM4), developed jointly by Los Alamos National Laboratory and NCAR. The script `ice.setup.csh` prepares the sea-ice model component for execution. To do so, the setup script defines the run environment, positions the sea-ice input data sets, then builds the dynamic sea-ice model by calling `gmake`.

If `ice.setup.csh` is unable to complete any of these tasks, it will abort with a non-zero error status. The `test.a1.run` script checks the error status and will halt if an error is detected.

### 12.1 Document the sea-ice setup script

```
#!/bin/csh -f
#=====
# CVS $Id: ice.setup.csh.tex,v 1.9 2002/10/10 19:17:01 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/ice.setup.csh.tex,v $
# CVS $Name: $
#=====
# ice.setup.csh: Preparing a CSM sea ice model, csim4, for execution
#
# (a) set environment variables, preposition input data files
# (b) create the namelist input file
# (b) build this component executable
#
# For help, see: www.cesm.ucar.edu/models/ccsm2.0.1
#=====
cat $0;$TOOLS/ccsm_checkenvs || exit -1          # cat this file, check envs
```

This first section documents the sea-ice setup script.

The first line of this section identifies this as a C-shell script. The `-f` option prevents the user's personalized `$HOME/.cshrc` file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The `cat` command combines two functions on one line. The `"cat $0"` command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then `$TOOLS/ccsm_checkenvs` writes the environment variables that have been set by `test.a1.run` into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.

## 12.2 Set the sea-ice model configuration flags

```

echo -----
echo a. set ice setup variables, get input files
echo -----

set PRESCRIBED_ICE      = .false.
set PRESCRIBED_ICE_CLIM = .false.
set OCEANMIXED_ICE     = .false.
set NCAT                = 5          # number of ice catagories

```

The first variables set in ice.setup.csh are flags that enable or disable the prescribed ice, climatological prescribed ice and the mixed-layer ocean options. For a fully coupled CCSM run, each of these flags will be set to .false.

- PRESCRIBED\_ICE** = **.false.** will result in the fully dynamical sea-ice model. Setting \$PRESCRIBED\_ICE to **.true.** will force the ice model to run in a thermodynamic-only state. Under these conditions, the ice extent and thickness are read in from a 19 year observational data file.
- PRESCRIBED\_ICE\_CLIM** = **.false.** disables the climatological cycling option for \$PRESCRIBED\_ICE = **.true.** \$PRESCRIBED\_ICE\_CLIM = **.true.** is a special case of PRESCRIBED\_ICE where the ice model continually cycles over 12 months of a climatological sea-ice data.
- OCEANMIXED\_ICE** = **.false.** disables the slab mixed-layer ocean model option in the sea-ice model. The slab mixed-layer ocean model option is generally used for sea-ice testing.

## 12.3 Acquire the sea-ice initial and boundary files

```

# Create directories for output hist/rest filenames
#-----

set HSTDIR = $EXEDIR/hist ; if !(-d $HSTDIR) mkdir -p $HSTDIR
set RSTDIR = $EXEDIR/rest ; if !(-d $RSTDIR) mkdir -p $RSTDIR
set INIDIR = $EXEDIR/init ; if !(-d $INIDIR) mkdir -p $INIDIR

# Calculate year used in new hist/rest filenames
#-----

if ($RUNTYPE == branch || $RUNTYPE == hybrid) then
  set DATEDASH = $REFDATE-00000
else
  set DATEDASH = $BASEDATE-00000
endif

```

Next, the directories for the history, restart and initial data are created if they don't already exist. Then the relevant date information is parsed into a form acceptable by CSIM based on the type of run being made.

```

# read in ice datasets
#-----

set ICEDATA = ice/csim4/
set RESTART = .true.
set PICE_DATA = " "
set OML_ICE_SST_INIT = .false.

rm -f data.domain.grid data.domain.kmt
$TOOLS/ccsm_getinput $ICEDATA/global_${ICE_GRID}.grid data.domain.grid || exit 2
$TOOLS/ccsm_getinput $ICEDATA/global_${ICE_GRID}.kmt data.domain.kmt || exit 2

```

**ICEDATA** = **ice/csim4/** defines the directory path (relative to \$CSM-DATA set in test.a1.run) for the ice initial and boundary data in the CCSM2 distribution.

**RESTART** = **.true.** instructs the ice model to read the names of the initial files from the restart pointer files.

**PICE\_DATA** = " " initializes the value of the variable containing the filename of the input data for the prescribed ice option to blank.

**OML\_ICE\_SST\_INIT** = **.false.** sets the flag controlling whether the SST values are initialized from the ocean fractional data

Next, files containing the sea-ice grid and land-mask information are copied into the \$EXEDIR directory.

```

if ($PRESCRIBED_ICE == .true.) then
  if ($RUNTYPE == startup) set RESTART = .false.
  if ($ICE_GRID == 'gx1v3') \
    set PICE_DATA = sst_clim_hurrell_gx1v3_020827.nc
  if ($ICE_GRID == 'gx3v4') \
    set PICE_DATA = sst_clim_hurrell_gx3v4_020827.nc

#-----Test for incompatible prescribed ice configurations-----

  if ($PICE_DATA =~ AMIP* ) then
    if ($PRESCRIBED_ICE_CLIM == .false. && $BASEDATE != 1977-01-01) then
      echo "Set BASEDATE in main script to 1977-01-01" ; exit -1
    endif
  endif
  if ($PICE_DATA =~ *hurrell* && $PRESCRIBED_ICE_CLIM == .false.) then
    echo "Set PRESCRIBED_ICE_CLIM == .true. in ice.setup.csh for Hurrell data" \
      ; exit -1
  endif

  $TOOLS/ccsm_getinput $ICEDATA/$PICE_DATA . || exit 99
endif

```

Here the actual initial and boundary files are positioned for the prescribed ice case. First, a specific BASEDATE of January 01 1977 is enforced when the ice model is not being run in climatological mode. Next, the RESTART flag is used to tell the model not to use the restart pointer files on a startup run to obtain the initial data. Then the time-varying boundary data sets are named and copied into place.

```

if ($RESTART == .true.) then
  if ($RUNTYPE == startup ) then
    set RSTFILE = iced.0001-01-01.${ICE_GRID}
    $TOOLS/ccsm_getinput $ICEDATA/$RSTFILE $RSTDIR/ || exit 2
    echo $RSTDIR/$RSTFILE >! $SCRIPTS/rpointer.ice
  else if ($RUNTYPE == branch || $RUNTYPE == hybrid) then
    set RSTFILE = $REFCASE.csim.r.$DATEDASH
    $TOOLS/ccsm_getfile $REFCASE/ice/rest/$RSTFILE $RSTDIR/ || exit 2
    echo $RSTDIR/$RSTFILE >! $SCRIPTS/rpointer.ice
  endif
endif
endif

```

For startup, hybrid and branch runs, the filenames for the initial data files are constructed and placed into the restart pointer files. ccsm\_getfile copies the initial data files into the restart directory. For a continuation run, nothing is done and the restart file names contained in the existing restart pointer files are used to continue the model run. Note, this script segment will be skipped for a startup run when \$PRESCRIBED\_ICE = .true. .

```

if ($OCEANMIXED_ICE == .true. ) then
  set POPFRC_DATA = pop_frc_${ICE_GRID}_010815.nc
  if ($ICE_GRID == 'gx3v4') \
    set POPFRC_DATA = pop_frc_gx3v4_020730.nc
  $TOOLS/ccsm_getinput $ICEDATA/$POPFRC_DATA oceanmixed_ice.nc || exit 99
  if ($RUNTYPE == startup) set OML_ICE_SST_INIT = .true.
  if ( $PRESCRIBED_ICE == .true.) then
    echo " Prescribed_ice cannot have Oceanmixed_ice -stop-" ; exit -1
  endif
endif
endif

```

Finally, for the slab ocean case, the initial data corresponding to the specified ice grid is copied in and the OML\_ICE\_SST\_INIT flag is set to initialize the SST's from the ocean data. If both OCEANMIXED\_ICE and PRESCRIBED\_ICE are set to .true., the script exits with an error status of -1.



## 12.4 Create the sea-ice namelist input file

```

echo -----
echo b. create the namelist input file
echo -----

cat << EOF >! ice_in
  &ice_nml
    runid          = '$CASE $CASESTR'
  , runtime       = '$RUNTYPE'
  , istep0        = 0
  , dt            = 3600.0
  , ndte         = 120
  , npt           = 99999
  , diagfreq     = 24
  , histfreq     = 'm'
  , dumpfreq     = 'y'
  , hist_avg     = .true.
  , restart      = $RESTART
  , print_points = .false.
  , kitd         = 1
  , kdyn         = 1
  , kstrength    = 1
  , evp_damping  = .false.
  , snow_into_ocn = .false.
  , advection    = 'mpdata2'
  , grid_type    = 'displaced_pole'
  , grid_file    = 'data.domain.grid'
  , kmt_file     = 'data.domain.kmt'
  , incond_dir   = '$INIDIR/'
  , incond_file  = '$CASE.csim.i.'
  , restart_dir  = '$RSTDIR/'
  , dump_file    = '$CASE.csim.r.'
  , history_dir  = '$HSTDIR/'
  , history_file = '$CASE.csim.h'
  , mss_dir      = '$MSSDIR/$MODEL/ '
  , mss_rtpd     = $MSSRPD
  , mss_pass     = '$MSSPWD'
  , mss_rmlf     = 0
  , prescribed_ice      = $PRESCRIBED_ICE
  , prescribed_ice_file = '$PICE_DATA'
  , prescribed_ice_climatology = $PRESCRIBED_ICE_CLIM
  , pointer_file        = '$SCRIPTS/rpointer.$MODEL'
  , oceanmixed_ice     = $OCEANMIXED_ICE
  , oceanmixed_ice_file = 'oceanmixed_ice.nc'
  , oceanmixed_ice_sst_init = $OML_ICE_SST_INIT
  , prntdiag_oceanmixed = .false.
/

```

```
&icefields_nml
  f_Fswdn   = .false.
, f_Flwdn   = .false.
, f_snow    = .false.
, f_rain    = .false.
, f_sst     = .false.
, f_sss     = .false.
, f_uocn    = .false.
, f_vocn    = .false.
, f_frzmlt  = .false.
, f_strltlx = .false.
, f_strltly = .false.
, f_Tref    = .false.
/
EOF
echo o contents of $MODEL.stdin: ; cat $MODEL.stdin ; echo ' '
```

This section constructs the input namelist that is used to control runtime operation of the sea-ice model. The namelist input file defines a wide range of parameters to control the behavior of the sea-ice model. The namelist input file, **ice.in**, is read by the sea-ice model on startup. Namelist input for the sea-ice model consists of text strings enclosed in quotes, integer and real numerical values and logicals.

The "cat" command uses the C-shell here-document option to create the file `$EXEDIR/ice.in` with all the settings being evaluated to the current values of the specified environment variables.

<b>&amp;ice_nml</b>	is the namelist group name, which matches the groupname defined within the sea-ice model.
<b>runid</b>	= ' <b>\$CASE \$CASESTR</b> ' (string) provides 80 characters to further describe this run. This description appears in the output logs and in the header data for the output data sets. <b>\$CASESTR</b> is set in the <b>\$SCRIPTS/test.a1.run</b> script.
<b>runtype</b>	= ' <b>\$RUNTYPE</b> ' (string) specifies the state in which this run is to begin. See the sample setup script section for discussion of this variable.
<b>istep0</b>	= <b>0</b> (integer) is the starting step number used to set the current elapsed time of the run on startup.
<b>dt</b>	= <b>3600.0</b> (real) sets the timestep for the ice transport and thermodynamics to 3600 seconds.
<b>ndte</b>	= <b>120</b> (integer) sets the number of ice model dynamics subcycles per model timestep to damp elastic waves.
<b>npt</b>	= <b>99999</b> (integer) sets the total number of timesteps. This is ignored in a coupled run where the flux coupler determines the stop point.
<b>diagfreq</b>	= <b>24</b> (integer) results in diagnostic printout being output at noon every 24 hours.
<b>histfreq</b>	= ' <b>m</b> ' (string) requests that history output be written monthly.
<b>dumpfreq</b>	= ' <b>y</b> ' (string) requests that restart output be written yearly. In a coupled run, this variable is ignored and instructions from the coupler are used.
<b>hist_avg</b>	= <b>.true.</b> (logical) defines the output history data as being averaged over the output period.
<b>restart</b>	= <b>\$RESTART</b> (logical) flags whether or not to read the restart data filenames from the restart pointer files.
<b>print_points</b>	= <b>.false.</b> (logical) flags whether or not to print out grid point data.
<b>kitd</b>	= <b>1</b> (integer) determines the type of itd conversions (0 = delta f, 1 = linear).
<b>kdyn</b>	= <b>1</b> (integer) determines the type of ice dynamics. Currently, only elastic-viscous-plastic dynamics (kdyn=1) is supported.
<b>kstrength</b>	= <b>1</b> (integer) instructs the ice to use Rothrock 1975 pressure formulation.
<b>evp_damping</b>	= <b>.false.</b> (logical) turns off the elastic-viscous-plastic ice damping.
<b>snow_into_ocn</b>	= <b>.false.</b> (logical) keeps ridging snow from being dumped into the ocean.
<b>advection</b>	= ' <b>mpdata2</b> ' (string) set the ice transport advection algorithm to a second order advection scheme using mpdata,
<b>grid_type</b>	= ' <b>displaced_pole</b> ' (string) sets the ice grid to displaced_pole rather than rectangular (default).
<b>grid_file</b>	= ' <b>data.domain.grid</b> ' (string) names the file containing the ice grid information.
<b>kmt_file</b>	= ' <b>data.domain.kmt</b> ' (string) names the file containing the land-mask information.
<b>incond_dir</b>	= ' <b>\$INIDIR/</b> ' (string) is the directory for snapshot initial conditions.
<b>incond_file</b>	= ' <b>\$CASE.csim.i.</b> ' (string) is the filename for snapshot initial conditions.

**restart\_dir** = '\$RSTDIR/' (string) is the directory for restart files.  
**dump\_file** = '\$CASE.csim.r.' (string) is the prefix of the filename for restart files.  
**history\_dir** = '\$HSTDIR/' (string) is the directory for history output files.  
**history\_file** = '\$CASE.csim.h' (string) is the prefix of the filename for output history files.  
**mss\_dir** = '\$MSSDIR/\$MODEL/' (string) ' is the NCAR Mass Storage System (MSS) directory for the output files.  
  
**mss\_rtpd** = \$MSSRPD (integer) is the retention period for MSS files.  
**mss\_pass** = '\$MSSPWD' (string) is the write password for MSS files.  
**mss\_rmlf** = 0 (integer) instructs the ice model not to remove files from disk after writing them to the MSS.  
  
**prescribed\_ice** = \$PRESCRIBED\_ICE (logical) is the flag for using the thermodynamic-only prescribed-ice option.  
**prescribed\_ice\_file** = '\$PICE\_DATA' (string) is the data file containing the prescribed ice data when using the thermodynamic only prescribed-ice option.  
**prescribed\_ice\_climatology** = \$PRESCRIBED\_ICE\_CLIM (logical) is the flag for using the climatology option to the thermodynamic only prescribed ice option.  
**pointer\_file** = '\$SCRIPTS/rpointer.\$MODEL' (string) gives the location of the restart pointer file.  
**oceanmixed\_ice** = \$OCEANMIXED\_ICE (logical) is the flag for using the slab mixed-layer ocean option.  
**oceanmixed\_ice\_file** = 'oceanmixed\_ice.nc' (string) is the input data for use by the slab mixed-layer ocean.  
**oceanmixed\_ice\_sst\_init** = \$OML\_ICE\_SST\_INIT (logical) controls whether the SST values are initialized from the ocean fractional data.  
**prntdiag\_oceanmixed** = .false. (logical) disables diagnostic printout from the slab mixed layer ocean.  
/  
**&icefields\_nml** marks the end of the ice\_nml group of namelist variables.  
/  
is an unused input namelist file.  
/  
marks the end of the icefields\_nml group of namelist variables.

## 12.5 Build the sea-ice model executable

The CSIM executable is built in this section. First the location of the source code is specified, then some resolution and processor tiling information is resolved. Once the source code Filepath and resolution information has been set, the CSIM executable is built with gmake.

```

echo -----
echo c. Build an executable in $OBJDIR
echo -----

cd $OBJDIR

# Filepath: List of source code directories (in order of importance).
#-----

\cat >! Filepath << EOF
$SCRIPTS/src.$MODEL
$CSMCODE/ice/csim4/src/source
$CSMSHR
EOF

```

The sea-ice model is built in the directory `$OBJDIR` to ensure that all the files involved with building the sea-ice model are in one directory.

The Filepath file contains the list of source-code directories from which to gather the input source code. This list will be used as the input to the gmake VPATH list.

The directories appearing in the Filepath file are listed in order of precedence, from most important to least important. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory listed, `$SCRIPTS/src.ice`, is typically used to hold modified sea-ice source code. If a directory in the filepath list is either empty or doesn't exist at all, no error will result.

```

# select resolution
#-----
if ( $GRID =~ *gx3*) set RES = 100x116
if ( $GRID =~ *hx1*) set RES = 384x320
if ( $GRID =~ *gx1*) set RES = 320x384
if !($?RES) echo "ERROR: ice can't deal with GRID = $GRID"
if !($?RES) exit -1

set NLON = `echo $RES | sed s/x.\*//`
set NLAT = `echo $RES | sed s/.\*//`

```

With Filepath set, the CCSM ice grid is parsed into integer longitude and latitude grid dimension values (NLON and NLAT) acceptable to the sea-ice model.

```

# Calculate processor tiling based on $NTASK
#-----
@ nlon = $NLON

setenv NY 1
if ( $GRID =~ *gx3*) then
  if ($NTASK > 7) setenv NY 2
else
  if ($NTASK > 16) setenv NY 2
endif

@ nx = $NTASK / $NY ; setenv NX $nx
@ rem = $nlon % $nx
if ($rem != 0) echo ERROR: NX must divide evenly into grid, $NLON,$NX
if ($rem != 0) exit -1

# Wipe and rebuild if NX or NY have changed
#-----
echo $NX $NY >! NXNY.new
cmp -s NXNY.old NXNY.new || rm $OBJDIR/*.o $EXEDIR/ice
echo $NX $NY >! NXNY.old

```

\$NX and \$NY specify the number of processors to assign to the latitude and longitude dimensions. This is designed to be automatic. More details on NX and NY can be found in the CSIM documentation.

The CSIM processor tiling is compiled into the model executable. Any changes to the number of processors assigned to CSIM will change the tiling, which requires recompiling the code. This section compares the currently requested tiling with the tiling specified during the last build; if the two tilings are different, a recompilation of CSIM is forced.

```

# Position new ice_model_size.F when needed
#-----
set ICESRC = $CSMCODE/ice/csim4/src/input_templates
cmp -s $ICESRC/ice_model_size.F.${RES}x${NCAT} ice_model_size.F || \
  cp $ICESRC/ice_model_size.F.${RES}x${NCAT} ice_model_size.F || exit 3

```

Predefined templates for the ice-model global domain size, ice categories and number of layers are copied into the current directory. If the current resolution and number of ice categories is different than what was used before, this routine will be recompiled.

```

# run make
#-----
#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c || exit 2
  if ($NTHRD > 1) setenv THREAD TRUE
  gmake -j 6 VPATH=Filepath MODEL=csim EXEC=$EXEDIR/$MODEL \
        NX=$NX NY=$NY THREAD=$THREAD \
        -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS || exit 2
else
  echo "BLDTYPE = $BLDTYPE"
endif

```

In this last section the sea-ice model is built using gmake.

The CCSM uses the gnumake (also known as “gmake”) tool to build the model executable. Each of the components setup scripts creates a list of source code directories from which to gather the input source code for that component. This list is called Filepath and will be used as the input to the gmake VPATH list. The file Filepath is written in each of the components \$OBJDIR directories.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.ice, is typically used to hold modified component source code. If a directory in the Filepath list is either empty or doesn't exist at all, no error will result. In general, the directories \$SCRIPTS/src.\$MODEL can be used to store locally modified source code. Each component script recognizes this directory as the top priority for finding source code.

First the makdep code is compiled. This utility program is called by the Makefile and checks for source code dependencies. This is done by seeing if any of the header or include files have been updated since the model was last built and ensures that the F90 modules are constructed in the proper order.

Once makdep is compiled, the GNU make program, gmake, is used to actually build the model. The -j 6 option uses 6 processors to build the model. The -f \$CSMBUILD/Makefile points to the generic CCSM Makefile while MACFILE=\$CSMBLD/Macros.\$OS points to the machine specific make options. MODEL identifies the component being built and VPATH points to the Filepath list. Finally, the actual executable to be built is \$EXEDIR/\$MODEL.

```

# document the source code used, cleanup $OBJDIR files
#-----

grep 'CVS' *. [hf]*
#gmake -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS mostlyclean

echo ' '
echo -----
echo End of setup shell script 'date'
echo -----

```

Once the executable is successfully built, the script documents the revision-control states of the source code and then exits with a 0 status.

## 13 The Land Model Setup Script: lnd.setup.csh

The land setup script, `scripts/test.a1/lnd.setup.csh`, is called by `test.a1.run` when the `lnd` model is selected in the `MODELS` environment variable array. The default land model is the Community Land Model (CLM) (<http://www.cgd.ucar.edu/tss/clm>) developed as a collaborative project between scientists in the Terrestrial Sciences Section of the Climate and Global Dynamics Division (CGD) at the National Center for Atmospheric Research (NCAR) and the CCSM Land Model Working Group. The script `lnd.setup.csh` prepares the land model component for execution by defining the run environment, positioning the land input data sets and building the dynamic land model by calling `gmake`.

If `lnd.setup.csh` is unable to complete any of these tasks, it will abort with a non-zero error status. The `test.a1.run` script checks the error status and will halt if an error is detected

### 13.1 Document the land setup script

```
#!/bin/csh -f
#=====
# CVS $Id: lnd.setup.csh.tex,v 1.10 2002/10/10 19:17:02 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/lnd.setup.csh.tex,v $
# CVS $Name: $
#=====
# lnd.setup.csh: Preparing a CCSM Common Land Model, clm2, for execution
#
# (a) set environment variables, preposition input data files
# (b) create the namelist input file
# (b) build this component executable
#
# For help, see: www.cesm.ucar.edu/models/ccsm2.0.1
#=====
cat $0;$TOOLS/ccsm_checkenvs || exit -1          # cat this file, check envs
```

This first section documents the land setup script.

The first line of this section identifies this as a C-shell script. The `-f` option prevents the user's personalized `$HOME/.cshrc` file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The `cat` command combines two functions on one line. The `"cat $0"` command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then `$TOOLS/ccsm_checkenvs` writes the environment variables that have been set by `test.a1.run` into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.



## 13.2 Set the land model configuration flags

```

echo -----
echo a. set environment variables, preposition input data files
echo -----

# lnd source code archive
#-----
set SRCDIR = $CSMCODE/lnd/clm2/src
set CAMLND_SHARE = $CSMCODE/camclm_share

```

First, the directories for the CLM source code and code which is shared with the CAM model are set.

```

cd $EXEDIR

# Set restart type
#-----
if ($RUNTYPE == 'startup') then
  set NSREST = 0
  set NREVSN = ""
  if ($GRID =~ T42* ) then
    set DATINIT = Ba.b19.clmi_00030101_00000.nc
    $TOOLS/ccsm_getinput lnd/clm2/inidata/csm/$DATINIT || exit 99
    if !(-f $DATINIT) exit 99
  else if ($GRID =~ T31* ) then
    set DATINIT = "b20.600a.clm2.i.0022-01-01-00000.nc"
    $TOOLS/ccsm_getinput lnd/clm2/inidata/csm/$DATINIT || exit 99
    if !(-f $DATINIT) exit 99
  else
    set DATINIT = ""
  endif
else if ($RUNTYPE == 'continue') then
  set NSREST = 1
  set DATINIT = ""
  set NREVSN = ""
else if ($RUNTYPE == 'branch' ) then
  set NSREST = 3
  set DATINIT = ""
  set NREVSN = ${REFCASE}.clm2.r.${REFDATE}-00000
  $TOOLS/ccsm_getfile $REFCASE/lnd/rest/$NREVSN || exit 99
  if !(-f $NREVSN) exit 99
else if ($RUNTYPE == 'hybrid') then
  set NSREST = 0
  set NREVSN = ""
  set DATINIT = ${REFCASE}.clm2.i.${REFDATE}-00000.nc
  $TOOLS/ccsm_getfile $REFCASE/lnd/init/$DATINIT || exit 99
  if !(-f $DATINIT) exit 99
endif

```

The current directory is set to `$EXEDIR` to ensure that all the files necessary to build and run the land model are in the correct directory.

Then the restart flag (`$NSREST`), the restart filename (`$NREVSN`) and the initial data file (`$DATINIT`) are set depending upon the type of run that was requested in the master `test.a1.run` script. If either an initial or a restart data file is needed, `ccsm_getinput` is called to position it in the current directory.

For `$RUNTYPE = 'startup'`, the restart flag (`$NSREST`) is set to 0, the branch restart filename (`$NREVSN`) is set to blank and the initial dataset (`$DATINIT`) is identified and positioned in the current directory.

For `$RUNTYPE = 'continue'`, the restart data come from the land restart files written previously. Here, the restart flag (`$NSREST`) is set to 1 and both the branch restart filename (`$NREVSN`) and the initial dataset (`$DATINIT`) are set to blank. The land model will internally generate the name of the required restart dataset by reading the restart pointer file.

`$RUNTYPE = 'branch'` is conceptually similar to a 'continue' run, except the restart file is explicitly named and the land model logic requires that the `$CASE` name be different from that of the reference `$CASE` name. To do this, the restart flag (`$NSREST`) is set to 3, the initial dataset (`$DATINIT`) is set to blank and the branch restart filename (`$NREVSN`) is identified and positioned in the current directory.

`$RUNTYPE = 'hybrid'` is similar to a 'startup' run. In this case, the name of the initial data set is generated from the reference case (`$REFCASE`) and reference date (`$REFDATE`) set in `test.a1.run`.

```
# Position the input data
#-----
set data_dir = lnd/clm2

if ($ATM_GRID == 'T85') set atmres = '256x128'
if ($ATM_GRID == 'T42') set atmres = '128x064'
if ($ATM_GRID == 'T31') set atmres = '096x048'
if ($ATM_GRID == 'T85') set DTIME = 600
if ($ATM_GRID == 'T42') set DTIME = 1200
if ($ATM_GRID == 'T31') set DTIME = 1800
if ($ATM_GRID == 'T85') set RTM_NSTEPS = 18
if ($ATM_GRID == 'T42') set RTM_NSTEPS = 9
if ($ATM_GRID == 'T31') set RTM_NSTEPS = 6

set surf_dat_date = ""
if ($atm_setup == atm && $ATM_GRID == 'T42') set surf_dat_date = ".080101"
set surf_dat = surface-data.${atmres}_atm.${OCN_GRID}_ocn${surf_dat_date}.nc
```

`$data_dir` specifies the directory path (relative to `$CSMDATA` set in `test.a1.run`) to the land model input data sets in the CCSM2 distribution. The atmosphere model resolution (`$atmres`), land model time step in seconds (`$DTIME`) and the number of model steps between river transport model calculations (`$RTM_NSTEPS`) are defined for the different supported resolutions. Finally, the name of the land surface dataset is based upon the atmosphere resolution, the type of atmosphere model and the ocean resolution.

A spun-up land initial dataset is supplied for use with the active atmosphere at T42 resolution. For other cases, the land model will try to do a "cold" start. During a cold start, the CLM is initialized from a generic state and will, over time, settle into an equilibrium state. The adjustment time for most CLM fields is around 5 years. However, it may take as long as 200 years for soil water content at the deepest layer to reach equilibrium.

```

$TOOLS/ccsm_getinput $data_dir/pftdata/pft-physiology          || exit 1
$TOOLS/ccsm_getinput $data_dir/rtmdata/rdirc.05              || exit 1
if ("$surf_dat" != "") then
  $TOOLS/ccsm_getinput $data_dir/srfdata/csm/$surf_dat        || exit 1
else
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_soicol_clm2.nc  || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_lanwat.nc       || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_glacier.nc     || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_urban.nc       || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_lai.nc         || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_pft.nc         || exit 1
  $TOOLS/ccsm_getinput $data_dir/rawdata/mksrf_soitex.10level.nc || exit 1
endif

set BASEDATE_NUM = `echo $BASEDATE | sed -e 's/-//g'` # remove "-"

```

A number of input datasets are required for the land model. First, Plant Functional Type (PFT) physiological constants and the River Transport Model (RTM) files are moved into the local directory.

If a surface dataset (\$surf\_dat) is defined, then it is acquired. Otherwise, the files in the rawdata directory are used to create a surface dataset at runtime.

### 13.3 Create the namelist input file

```
# Create the input parameter namelist file
#-----
cat >! lnd.stdin << EOF
&clmexp
rtm_nsteps      = $RTM_NSTEPS
caseid          = '$CASE'
ctitle          = '$CASE $CASESTR'
nsrest          = $NSREST
start_ymd       = $BASEDATE_NUM
start_tod       = 0
nelapse         = -9999
dtime           = $DTIME
irad            = -1
csm_doflxave    = .true.
nrevsn          = '$NREVSN '
finidat         = '$DATINIT '
fsurdat         = '$surf_dat'
fpftcon         = 'pft-physiology'
frivinp_rtm     = 'rdirc.05'
mksrf_fvegtyp   = 'mksrf_pft.nc'
mksrf_fsoitex   = 'mksrf_soitex.10level.nc'
mksrf_fsoicol   = 'mksrf_soicol_clm2.nc'
mksrf_flanwat   = 'mksrf_lanwat.nc'
mksrf_fglacier  = 'mksrf_glacier.nc'
mksrf_furban    = 'mksrf_urban.nc'
mksrf_flai      = 'mksrf_lai.nc'
mss_wpass       = '$MSSPWD'
hist_nhtfrq     = 0
hist_crtinic    = 'YEARLY'
mss_irt         = $MSSRPD
rpntpath        = '$SCRIPTS/rpointer.$MODEL'
/
EOF
```

In the namelist input file, a wide range of parameters are set to control the behavior of the land model. The namelist input file, **clmexp**, is a text file that is read by the land model. This section constructs the input namelist that is used to control runtime operation of the land model.

The "cat" command uses the Unix here-document option to create the file `$EXEDIR/clmexp` with all the settings being evaluated to the current values of the specified environment variables.

<b>&amp;clmexp</b>	is the namelist group name, which matches the groupname defined within the land-surface model.
<b>rtm_nsteps</b>	= <b>\$RTM_NSTEPS</b> (integer) sets the number of timesteps between river runoff calculations.
<b>caseid</b>	= <b>'\$CASE'</b> (string) sets a unique, 16-character, test string used to identify this run. The CASE variable is set in \$SCRIPTS/test.a1.run script and is used extensively in the CLM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX file-naming characters such as letters, numbers, underscores, dashes, commas or periods.
<b>ctitle</b>	= <b>'\$CASE \$CASESTR'</b> (string) provides 80 characters to further describe the run. This description appears in the output logs and in the header data for the output data sets. CASESTR is set in the run script.
<b>nsrest</b>	= <b>\$NSREST</b> (integer) specifies the state in which the run is to be started. nsrest settings 0,1,3 map into the CCSM variables (startup/hybrid, continue and branch).
<b>start_ymd</b>	= <b>\$BASEDATE_NUM</b> (integer) specifies the "basedate" for the run. This date serves as the baseline from which the current step number is calculated.
<b>start_tod</b>	= <b>0</b> (integer) sets the starting time of day for the run in seconds.
<b>nelapse</b>	= <b>-9999</b> (integer) indicates that the flux coupler will control the ending timestep for this model run.
<b>dtime</b>	= <b>\$DTIME</b> (integer) sets the timestep, in seconds, for the land model. The timestep is dependent on the resolution of the model.
<b>irad</b>	= <b>-1</b> (integer) sets the iteration frequency for shortwave radiation calculations. If the number is positive, then the frequency is specified in model timesteps. If the number is negative, then the frequency is specified in model hours.
<b>csm_doflxave</b>	= <b>.true.</b> (logical) instructs the land model to only communicate with flux coupler on albedo calculation timesteps.
<b>nrevsn</b>	= <b>'\$NREVSN'</b> (string) is the optional name of the branch restart dataset to be read if this is a branch run. This dataset is a binary file containing the exact state of a previous run.
<b>fnidat</b>	= <b>'\$DATINIT'</b> (string) sets the name of the initial data file. If \$DATINIT is blank, then the land model is initialized from a "cold start" condition.
<b>fsurdat</b>	= <b>'\$surf_dat'</b> (string) is name of the land surface dataset. If \$surf_data is blank, then the land model surface dataset at the model resolution is created at runtime from a "cold start" condition.
<b>fpftcon</b>	= <b>'pft-physiology-vegdyn-cleanup-ratio'</b> (string) identifies the Plant Functional Type (PFT) physiological constants input dataset.
<b>frivinp_rtm</b>	= <b>'rdirc.05'</b> (string) is the name of the River Transport Model (RTM) input dataset.
<b>mksrf_fvegtyp</b>	= <b>'mksrf_pft.nc'</b> (string) is the vegetation type data file name.
<b>mksrf_fsoitex</b>	= <b>'mksrf_soitex.10level.nc'</b> (string) is the soil texture data file name.
<b>mksrf_fsoicol</b>	= <b>'mksrf_soicol_clm2.nc'</b> (string) is the soil color data file name.
<b>mksrf_flanwat</b>	= <b>'mksrf_lanwat.nc'</b> (string) is the inland water data file name.
<b>mksrf_fglacier</b>	= <b>'mksrf_glacier.nc'</b> (string) is the glacier data file name.
<b>mksrf_furban</b>	= <b>'mksrf_urban.nc'</b> (string) is the urban data file name.

**mksrf\_flai** = 'mksrf\_lai.nc' (string) is the leaf area index data file file name.  
**mss\_wpass** = '\$MSSPWD' (string) sets the write password for any files written to the NCAR MSS.  
**hist\_nhtfrq** = 0 (integer) is the history tape output interval (+ = iterations, - = hours, 0=monthly average)  
**hist\_crtinic** = 'YEARLY' (string) outputs a land initial condition file if set to 'MONTHLY' or 'YEARLY',  
**mss\_irt** = \$MSSRPD (integer) specifies the mass store retention period for output history and restart files when running at NCAR. If mss\_irt is set to 0, the output history and restart files are not written to the Mass Storage System.  
**rpntpath** = '\$SCRIPTS/rpointer.\$MODEL' (string) is the filename of the restart pointer file. The restart pointer file is a one line text file containing the name of the CLM restart files needed to continue the run.  
 / marks the end of the clmexp namelist input variables.  
**EOF** marks the end of the Unix here document referenced by the cat command near the start of this section.

### 13.4 Create the land model executable

The rest of the lnd.setup.csh builds the CLM executable. First the location of the source code is specified, then various resolution and configuration information is put into header files. Once the source code Filepath and header files have been created, the CLM executable is built with gmake.

```

echo '-----'
echo b. Build an executable in $OBJDIR
echo '-----'

cd $OBJDIR

# build Filepath: List of source code directories (in order of importance)
# (update only if new or changed)
#-----
\cat >! .tmp << EOF; cmp -s .tmp Filepath || mv -f .tmp Filepath
$SCRIPTS/src.lnd
$SRCDIR/main
$SRCDIR/biogeophys
$SRCDIR/ecosysdyn
$SRCDIR/riverroute
$SRCDIR/biogeochem
$SRCDIR/mksrfddata
$CAMLND_SHARE
$CSMSHR
$CSMCODE/utils/timing
EOF

```

The land-surface model is built in the directory \$OBJDIR to ensure that all the files involved with building the land-surface model are in one directory.

The Filepath file contains the list of source-code directories from which to gather the input source code. This list will be used as the input to the gmake VPATH list.

The directories appearing in the Filepath file are listed in order of precedence, from most important to least important. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory listed, \$SCRIPTS/src.lnd, is typically used to hold modified land-model source code. If a directory in the filepath list is either empty or doesn't exist at all, no error will result.

```
# build preproc.h: land model dimensions
# (update only if new or changed)
#-----
if ($GRID =~ T85* ) set LONLAT = ( 256 128 )
if ($GRID =~ T42* ) set LONLAT = ( 128 64 )
if ($GRID =~ T31* ) set LONLAT = ( 96 48 )

\cat >! .tmp << EOF; cmp -s .tmp preproc.h || mv -f .tmp preproc.h
#ifndef PREPROC_SET
#define PREPROC_SET
#define LSMLON $LONLAT[1]
#define LSMLAT $LONLAT[2]
#endif
EOF
```

With Filepath set, the CCSM land grid is parsed into integer longitude and latitude grid dimension values (LSMLON and LSMLAT) acceptable to the land model. These dimension values are inserted into the preproc.h header file.

```
# build misc.h
# (update only if new or changed)
#-----
set spmd = "#undef SPMD"
if ($NTASK > 1) set spmd = "#define SPMD"

\cat >! .tmp << EOF; cmp -s .tmp misc.h || mv -f .tmp misc.h
#ifndef MISC_SET
#define MISC_SET
$spmd
#define COUP_CSM
#define RTM
#endif
EOF
```

The misc.h header file sets options for automatically building the coupled version of the model (COUP\_CSM), building a single-task or multi-task version of the code and using the River Transport Model (RTM). Note, this document only discusses the coupled version. See the CLM model User's Guide (<http://www.cesm.ucar.edu/models/ccsm2.0.1/UsersGuide/UsersGuide>). for information on running the land model as a "standalone" model.

```

# Create the executable
#-----
#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

if ($BLDDTYPE == 'true') then
  set echo on
  cc -o makdep $CSMBLD/makdep.c
  if ($NTHRD > 1) setenv THREAD TRUE
  gmake -j 6 -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS MODEL=clm \
          VPFILe=Filepath EXEC=$EXEDIR/lnd || exit 2
else
  echo "BLDDTYPE = $BLDDTYPE"
endif
wait

```

The land model executable is only built if \$BLDDTYPE is true.

The CCSM uses the gnumake (also known as “gmake”) tool to build the model executable. Each of the components setup scripts creates a list of source code directories from which to gather the input source code for that component. This list is called Filepath and will be used as the input to the gmake VPATH list. The file Filepath is written in each of the components \$OBJDIR directories.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.lnd, is typically used to hold modified component source code. If a directory in the Filepath list is either empty or doesn't exist at all, no error will result. In general, the directories \$SCRIPTS/src.\$MODEL can be used to store locally modified source code. Each component script recognizes this directory as the top priority for finding source code.

First the makdep code is compiled. This utility program is called by the Makefile and checks for source code dependencies. This is done by seeing if any of the header or include files have been updated since the model was last built and ensures that the F90 modules are constructed in the proper order.

Once makdep is compiled, the GNU make program, gmake, is used to actually build the model. The -j 6 option uses 6 processors to build the model. The -f \$CSMBUILD/Makefile points to the generic CCSM Makefile while MACFILE=\$CSMBLD/Macros.\$OS points to the machine specific make options. MODEL identifies the component being built and VPFILe points to the Filepath list. Finally, the actual executable to be built is \$EXEDIR/\$MODEL.



## 14 The Data Model Setup Scripts: d\*\*\*.setup.csh

Any of the CCSM models can be replaced by a simple data component which simply reads previously calculated data for that component and supplies these data to the coupler. The data components were previously an integral step in the CSM1 model spin-up procedure, but now the CCSM2 spin-up involves starting the full set of active models from climatological conditions, without using the data components. While the data components now play a lesser role in making a CCSM run, they are valuable tools for testing and debugging. For this reason, they are included in the distribution package as examples of ways to supply existing data to the CCSM.

To use a data component in place of the active component, replace the name of the active component (i.e. **atm** for the atmosphere) with the data-model equivalent in the proper place in the \$SETUPS array definition in \$SCRIPTS/test.a1.run script. Also, \$NTHRDS and \$NPROCS should both be set to 1. This may require a change in the batch queue setting associated with the number of threads and processes (task\_geometry on the IBM SP, mpp\_p on the SGI and nodes:ppn on the Compaq).

All of the data component scripts follow a similar pattern, so only the data atmosphere script will be described here.

```

#!/bin/csh -f
#=====
# CVS $Id: data.models.tex,v 1.8 2002/10/10 19:17:01 southern Exp $
# CVS $Source: /fs/cgd/csm/models/CVS.REPOS/shared/ccsm/doc/UsersGuide/data.models.tex,v $
# CVS $Name: $
#=====
# datm.setup.csh: Prepare a CSM data atmosphere component, datm5, for execution
#           For help, see: www.ccsm.ucar.edu/models/ccsm2.0.1
#=====
cat $0;$TOOLS/ccsm_checkenvs || exit -1           # cat this file, check envs

```

The first line of this section identifies this as a C-shell script. The "-f" option prevents the user's personalized \$HOME/.cshrc file from being executed to avoid introducing aliases that could adversely affect the operation of this script.

The CVS lines document the revision control version of this script.

The echo lines document the purpose of this script. These output from the echo commands will appear in the component log files.

The cat command combines two functions on one line. The "cat \$0" command prints a copy of the entire setup script into the output log file in order to document the exact options set by this script. Then \$TOOLS/ccsm\_checkenvs writes the environment variables that have been set by test.a1.run into the same output log file. If any of the required environment variables are not set, the setup script will exit with an error status of -1.

```

echo -----
echo a. set environment variables, preposition input data files
echo -----

if ( $GRID =~ T31*) then
# set DATA_DIR      = mss:/KAUFF/csm/f20.007/atm/hist      # model does mss read
# set DATA_DIR      = cp:$CSMDATA/atm/datm5                # model does local copy
  set DATA_DIR      = null:/doesnt/matter                 # must preposition data
  set DATA_FNAME    = f20.007.cam2.h1.yyyy-01-01-00000.nc # generic file name
  set DATA_FNAME1   = f20.007.cam2.h1.0006-01-01-00000.nc # specific file name
  $TOOLS/ccsm_getinput atm/datm5/$DATA_FNAME1             # prepositioning util
else if ( $GRID =~ T42*) then
# set DATA_DIR      = mss:/KAUFF/csm/f20.006/atm/hist      # model does mss read
# set DATA_DIR      = cp:$CSMDATA/atm/datm5                # model does local copy
  set DATA_DIR      = null:/doesnt/matter                 # must preposition data
  set DATA_FNAME    = f20.006.cam2.h1.yyyy-01-01-00000.nc # generic file name
  set DATA_FNAME1   = f20.006.cam2.h1.0006-01-01-00000.nc # specific file name
  $TOOLS/ccsm_getinput atm/datm5/$DATA_FNAME1             # prepositioning util
else
  echo "ERROR: unsupported grid = $GRID"
  exit -1
endif

set RUN_TYPE = $RUNTYPE
if ($RUNTYPE == startup) set RUN_TYPE = initial
if ($RUNTYPE == hybrid) set RUN_TYPE = initial

```

\$DATA\_DIR points to specifies the directory path to the input data sets. The first DATA\_DIR setting which is commented out with a # in the 1st column, shows a local NCAR method for pointing to a collection of T42 input data. The second setting illustrates how to copy the data from the \$CSMDATA area. The third setting is another local NCAR methods for acquiring the T31 input data.

The set of four commented out lines (using the C-shell # symbol) is another method which simply places the required data in the execution directory. Since the data component will always look in the local directory first before trying to find the data in a remote spot, this is the most straightforward method. The disadvantage of this is that the names of the input files must be specifically set, while the previous methods will position the files as they are needed.

The \$RUNTYPE is parsed for use in the data component. For \$RUNTYPE = branch or continue, no translation is needed. However both \$RUNTYPE = startup and hybrid are interpreted as an 'initial' run for the data component.

## 14.1 Create the namelist input file

```

echo -----
echo b. create the namelist input file
echo -----

cat >! $MODEL.stdin << EOF
&inparm
  case_name = '$CASE'
  case_desc = '$CASE $CASESTR'
  rest_type = '$RUN_TYPE'
  ncpl      = 24
  data_dir  = '$DATA_DIR'
  data_fname = '$DATA_FNAME'
  data_year0 = 0006
  data_nyear = 1
  data_oyear = 1
  flux_albfb = 1
/
EOF

```

This section constructs the input namelist which is used to control runtime operation of the data component.

The "cat" command creates the file \$EXEDIR/\$MODEL.stdin with all the settings being evaluated to the current values of the specified environment variables.

<b>&amp;inparm</b>	is the namelist group name, which matches the groupname defined within the data component.
<b>caseid</b>	= <b>\$CASE</b> (string) is a unique string (16 characters or less) that is used to identify this run. The CASE variable is set in \$SCRIPTS/test.a1.run and is used extensively in the CCSM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX file-naming characters such as letters, numbers, underscores, dashes, commas or periods.
<b>ctitle</b>	= <b>'\$CASE \$CASESTR'</b> (string) provides 80 characters to further describe this run. This description appears in the output logs and in the header data for the output data sets. CASESTR is set in the \$SCRIPTS/test.a1.run script.
<b>rest_type</b>	= <b>'\$RUN_TYPE'</b> (string) specifies the state in which the run is to be started. rest_type settings initial, branch and continue map into the CCSM variables (startup/hybrid branch and continue).
<b>ncpl</b>	= <b>24</b> (integer) sets the coupling frequency to 24 times per day.
<b>data_dir</b>	= <b>'\$DATA_DIR'</b> (string) supplies the location of the input data directory to the code.
<b>data_year0</b>	= <b>0006</b> (integer) sets the first year of the input data to be year 6.
<b>data_nyear</b>	= <b>1</b> (integer) sets the total number of years to cycle the input data over to 1.
<b>data_oyear</b>	= <b>1</b> (integer) defines the coupler year corresponding to data_year0. This allows data with arbitrary dates to be input.
<b>flux_albfb</b>	= <b>1</b> (integer) enables the optional albedo feedback calculation.

The data ocean component (docn) namelist has been modified to provide more functionality. The following variables have been removed: data\_miss data\_year0, data\_rmlf, data\_nyear and data\_oyear.

The following namelist inputs appear in the data-ocean component.

**domain\_file** = '\$FILE\_DOMAIN' (string) is the name name of domain netcdf file.  
**data\_file** = '\$FILE\_SST' (string) is the input sea-surface temperature file.  
**data\_form** = '\$DATA\_FORM' (string) describes the time configuration of the data either 'annual' or 'multiyear' DEFAULT is 'annual'  
**data\_sstname** = '\$DATA\_SSTNAME' (string) is the name of the sst field on input data file. DEFAULT is 'T'.  
**data\_lonname** = 'xc' (string) name of longitude coordinate on input data file. DEFAULT is 'xc' (can be one dimensional, or two dimensional for non-rectilinear coordinates)  
**data\_latname** = 'yc' (string) name of latitude coordinate on input data file, DEFAULT is 'yc' (can be one dimensional, or two dimensional for non-rectilinear coordinates).

```

echo -----
echo c. Build an executable in $OBJROOT
echo -----

# Filepath: List of source code directories (in order of importance).
#-----

cd $OBJDIR

\cat >! Filepath << EOF
$SCRIPTS/src.$MODEL
$CSMCODE/atm/datm5
$CSMSHR
EOF

# Check to see if a new dims.h is needed.
#-----

set dims = $CSMCODE/atm/datm5/dims.h.$ATM_GRID
if (-f $SCRIPTS/src.$MODEL/dims.h) set dims = $SCRIPTS/src.$MODEL/dims.h
cmp -s $dims dims.h || cp $dims dims.h || exit 3

```

The data component executable is built in \$OBJDIR.

The filepath is the list of source code directories from which to gather the input source code. This list will be used as the input to the gmake VPATH list.

The filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.\$MODEL, is typically used to hold modified source code. If a directory in the filepath list is either empty or doesn't exist at all, no error will result.

Next, the file containing the dimensions of the data component is positioned in \$OBJDIR if it is different than the existing dimensions file.

```

# run make
#-----
#--- run machine dependent commands (i.e. modules on SGI).
if (-f $TOOLS/modules.$OS.$MACH) source $TOOLS/modules.$OS.$MACH || exit 1

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c                               || exit 2
  if ($NTHRD > 1) setenv THREAD TRUE
  gmake -j 6 VPFIL=Filepath MODEL=datm EXEC=$EXEDIR/$MODEL \
    -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS          || exit 2
else
  echo "BLDTYPE = $BLDTYPE"
endif

```

In this last section the data component is built using gmake if \$BLDTYPE is set to 'true'. The CCSM uses the gnumake (also known as "gmake") tool to build the model executable. Each of the components setup scripts creates a list of source code directories from which to gather the input source code for that component. This list is called Filepath and will be used as the input to the gmake VPATH list. The file Filepath is written in each of the components \$OBJDIR directories.

The Filepath directories are listed in order of precedence. If a file is found in more than one of the directories listed in Filepath, the version of the file found in the directory listed first will be used to build the code. The first directory, \$SCRIPTS/src.cpl, is typically used to hold modified coupler source code. If a directory in the Filepath list is either empty or doesn't exist at all, no error will result. In general, the directories \$SCRIPTS/src.\$MODEL can be used to store locally modified source code. Each component script recognizes this directory as the top priority for finding source code.

First the makdep code is compiled. This utility program is called by the Makefile and checks for source code dependencies. This is done by seeing if any of the header or include files have been updated since the model was last built and ensures that the F90 modules are constructed in the proper order.

Once makdep is compiled, the GNU make program, gmake, is used to actually build the model. The -j option specifies the number of processors to use to build the model. The -f \$CSMBUILD/Makefile points to the generic CCSM Makefile while MACFILE=\$CSMBLD/Macros.\$OS points to the machine specific make options. MODEL identifies the component being built and VPFIL points to the Filepath list. Finally, the actual executable to be built is \$EXEDIR/\$MODEL.

At this point, control is returned to test.a1.run.

## 15 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. See also `ccsm_archive`.
- **\$ARCH** - local hardware architecture, typically IBM, SGI, CPQ, etc.
- **\$ARCROOT** - full path to the archival directory. See also **archive** and `ccsm_archive`.
- **\$CASE** - experiment or case name.
- **ccsm\_archive** - a script that archives model output. See also **archive**.
- **ccsm\_joe** - a CASE-dependent CCSM resource file containing the environment variable values used during the CCSM run. **ccsm\_joe** is written by the main run script each time the script is executed. Used as a debugging tool and resource file.
- **\$CSMDATA** - full path to the inputdata directory.
- **\$CSMEXE** - full path to the top directory where CCSM is going to run.
- **\$CSMROOT** - full path to the top directory of the source code release.
- **harvester** - a phase of the CCSM production process in which model output is moved from local disk to the local long-term storage system. The harvester script is named `SCRIPTS/$CASE.har`.
- **\$MACH** - machine name, typically blackforest, seaborg, nirvana, etc.
- **main run script** - The C-shell scripts which runs the CCSM. In the CCSM distribution, the example run script is `SCRIPTS/test.a1.run`.
- **\$MODELS** - always “atm lnd ice ocn cpl” currently.
- **\$NTASK** - the number of MPI tasks for each model, set in the main run script.
- **\$NTHRD** - the number of OpenMP threads for each MPI task for each model. Set in the main run script.
- **\$OS** - local operating system, typically AIX, IRIX64, etc.
- **\$RUNTYPE** - environment variable set in the main run script to describe the type of the run, can be startup, branch, hybrid, or continue.
- **\$SCRIPTS** - full path to the scripts directory (often this is `$CSMROOT/scripts`).
- **\$SETUPS** - the version of the models to be used in the main run script. typically `cpl, lnd/dlnd, ice/dice, ocn/docn` and `atm/datm/latm`.
- **\$SITE** - location, typically ncar, lanl, nersc, etc. The **main run script** attempts to set this automatically based on known site. **\$SITE** is used by the Gnu Makefile to enable site specific commands.
- **test.a1.har** - the harvester script called from the **main run script**. See also **archive** and **harvester**.
- **test.a1.run** - the name of the **main run script** in the example test.a1 case.