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

CCSM2.0 User's Guide

Lawrence Buja and Tony Craig

www.ccsm.ucar.edu

Build date: June 18, 2002
CVS tag `$Name: doc_a020618 $`

Contents

1 Introduction

The Community Climate System Model (CCSM) is a coupled climate model for simulating the earths climate system. Composed of four separate model components simultaneously simulating the earths atmosphere, ocean, land surface and sea-ice, and one central coupler component, the CCSM allows researchers to conduct fundamental research into the earths 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 May 17 2002 CCSM2.0 release specifically supports the high-resolution (T42 atmosphere/land and gx1v2 ocean/sea-ice) for IBM SPII 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.ccsm.ucar.edu .

1.1 The CCSM component models

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

cpl	The Coupler
atm	The Atmospheric component
ocn	The Ocean component
ice	The Sea-ice component
lnd	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.

1.2 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.ccsm.ucar.edu/models/ccsm2.0

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-degree (gx3) horizontal resolutions. The POP web page can be found at www.ccsm.ucar.edu/models/ccsm2.0/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, which are identical to those used by the POP ocean model. The CSIM web page is www.ccsm.ucar.edu/models/ccsm2.0/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

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.ccsm.ucar.edu/models/ccsm2.0/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 release page: www.ccsm.ucar.edu/models/ccsm2.0

1.3 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 gx3 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: 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, gx3
- 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_gx3		*								
T62_gx1v3						*				*

* indicates tested configuration

2 CCSM2.0 Quick Start Guide

This section briefly describes how to download the CCSM2.0 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 website: www.ccsm.ucar.edu/models/ccsm2.0
- Untar the source code into your home directory (or similar)
- Untar the input datasets into a large disk area
- Create a new case uby sing the CCSM graphical user interface (GUI) found in the $HOME/ccsm2.0/scripts/gui_run directory or by manually copying and reconfiguring the example CCSM scriptsin the $HOME/ccsm2.0/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.

- OS: IBM AIX or SGI IRIX64
- Compilers: Fortran 90, C
- Tools: gunzip, gnumake, Perl5 or greater, Perl/Tk for gui_run
- Permanent disk space
- Temporary disk space
- Libraries: MPI an, netCDF
- Input data : 0.2 GB for T31_gx3 grid, 0.8 GB for T42_gx1v3 grid
- CCSM2 source code: 35 MB

2.2 Downloading and untarring the CCSM2.0 distribution

CCSM2.0 is available via the web from: **www.cgd.ucar.edu/csm/models/ccsm2.0**

The CCSM2.0 distribution consists of the following files:

- **ccsm2.0.tar.gz** CCSM2.0 source codes, run scripts and documentation
- **ccsm2.0.inputdata.T31_gx3.tar** CCSM2.0 input data for T31_gx3 resolution
- **ccsm2.0.inputdata.T42_gx1v3.tar** CCSM2.0 input data for T42_gx1v3 resolution
- **ccsm2.0.inputdata.T62.tar** CCSM2.0 latm data for T62 resolution
- **ccsm2.0.inputdata.CLM_SPINUP_FILES.tar** are optional files needed to generate a CLM surface dataset

To uncompress and untar the file ccsm2.0.tar.gz, use the Unix gunzip and tar commands:

```
gunzip -c ccsm2.0.tar.gz | tar -xf -
```

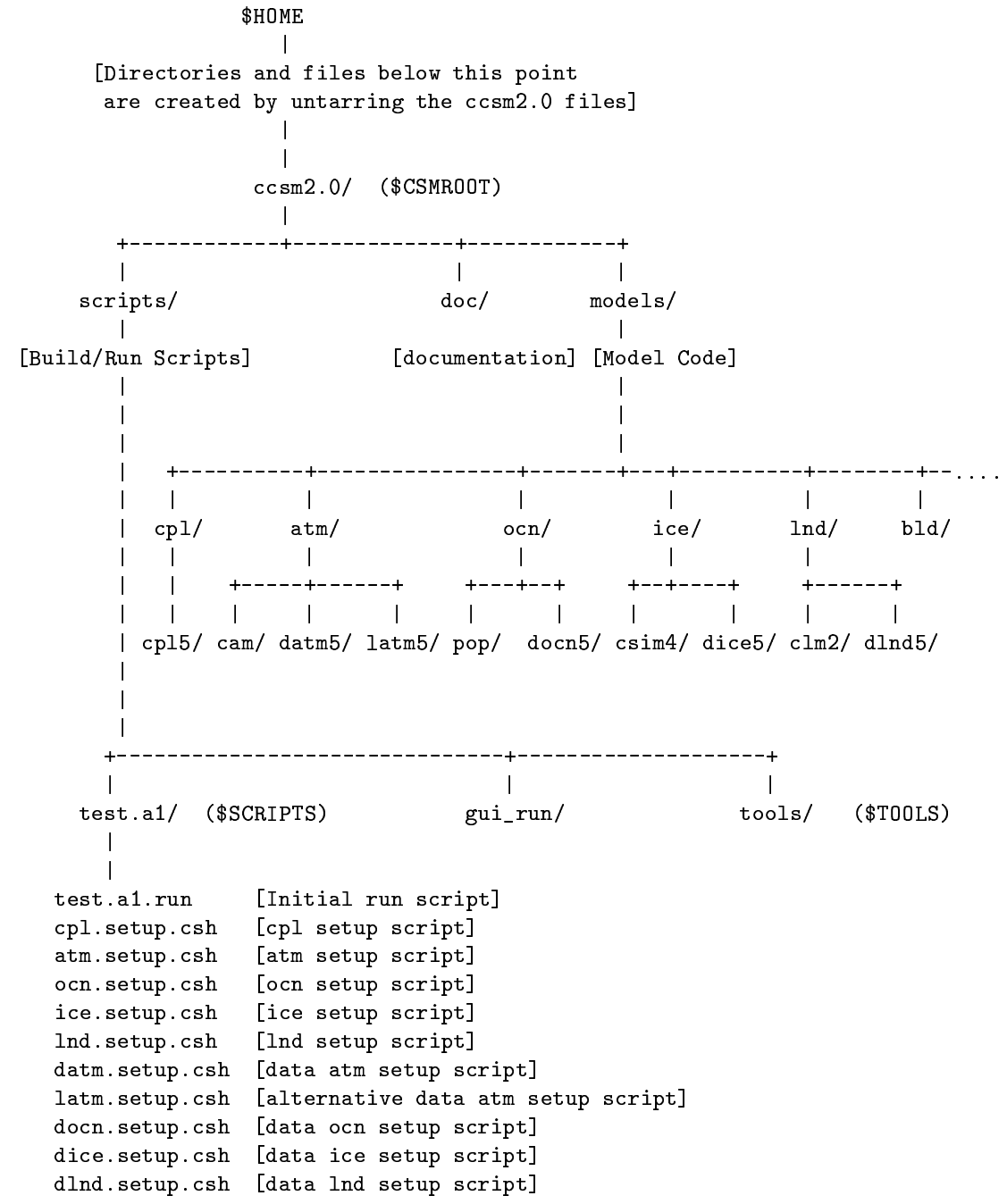
To untar the data files:

```
tar -xf ccsm2.0.inputdata.T42_gx1v3.tar
```

By default, the CCSM2.0 distribution is configured to run on the NCAR IBM SP blackforest under the directory /ptmp/$LOGNAME/$CASE

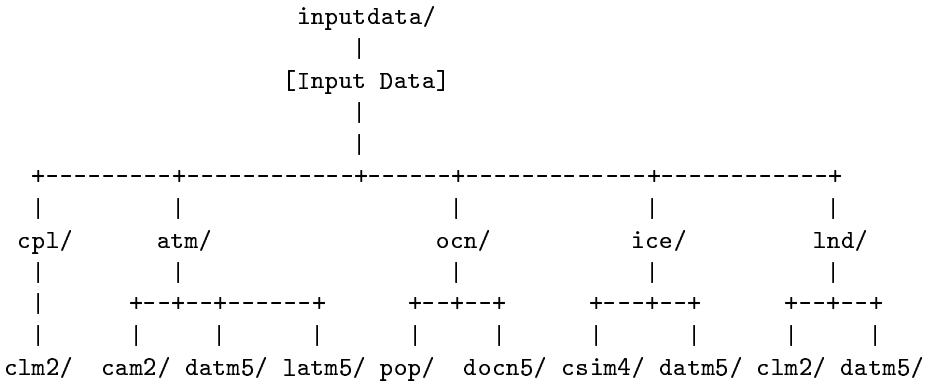
2.3 The CCSM source code

Assuming that the CCSM2.0 distribution has been untarred under the directory $HOME/, the top levels of the resulting directory tree are:



2.4 Input datasets

The directory tree for the tarfile containing the CCSM2.0 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 main 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 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 vertical levels in 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 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 into the new case directory.

```
cd /home/\$LOGNAME/ccsm2.0/scripts
mkdir newcase
cp test.a1/* newcase/
```

- Modify the main run script

```
cd /home/\$LOGNAME/ccsm2.0/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
  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 ARCROOT" to the local directory for archiving model output
```

- Run the script
  submit /home/$LOGNAME/ccsm2.0/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 writes monthly averaged history files for all components in netCDF format. CCSM2.0 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 CCSM distribution, this file is $HOME/ccsm2.0/scripts/test.a1.run. 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 lnd) 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 1.7 2002/05/13 20:07:10 southern Exp $
# CVS Source:  $
# CVS Name: ccsm2_0_beta47 $
```

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)}
# @ 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: ute ------------
# 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
#-------------------------------------------------------------------------
```

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 queueing 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.

3.1.2 Define the common run environment

```
echo ------------------------------------------------------------------------
echo  b.  Set env variables available to model setup scripts
echo ------------------------------------------------------------------------

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

setenv CASE        test.a1                      # case name
setenv GRID        T42_gx1v3                    # T42_gx1v3 or T31_gx3
setenv RUNTYPE     startup                 # startup, continue, branch, hybrid
setenv SETBLD      auto                         # auto, true, false
setenv BASEDATE    0001-01-01                   # initial start date yyyy-mm-dd
```

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

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.

CASE (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.

GRID (string) specifies the CCSM horizontal grid. The format is atm/lnd_ocn/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_gx3 for all configurations other than latm, and T62_gx1v3 for latm.

RUNTYPE (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.

SETBLD (string) controls whether or not the model executable is built. If **SETBLD = true**, all of the CCSM component executables will be rebuilt if gmake determines it is needed. For **SETBLD = false**, the component executables will not be rebuilt. For **SETBLD = auto** the components will not be rebuilt if **RUNTYPE** is continue. For all other RUNTYPE parameters, **SETBLD = auto** will invoke gmake. This ensures that the same component executables are used during the entire integration when running production runs using **RUNTYPE = continue**.

BASEDATE (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 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**.

```
setenv CASESTR "fully coupled $GRID test"        # short descriptive text string
setenv MSSDIR   mss:/$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
```

CASESTR (string) is designed to hold a short description of this CASE. CASESTR will appear in the model output and in the global attributes of the output data files.

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 archiver and harvester scripts in the $SCRIPTS directory (see below) can be 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 CSMROOT  /fs/cgd/home0/$LOGNAME/ccsm2.0   # root directory of source
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/utils            # Util directory
setenv CSMSHR   $CSMROOT/models/csm_share        # shared code dir
setenv CSMBLD   $CSMROOT/models/bld              # makefiles are here
setenv CSMDATA  /fs/cgd/csm/inputdata            # base dir for input data
setenv LID      "`date +%y%m%d-%H%M%S`"          # time-stamp/file-ID string
```

CSMROOT (string) defines the root directory of the CCSM code distribution directory tree. The model source code, scripts and documention are located underneath this directory. In the default case, all of the following environment variables are be derived from CSMROOT.

SCRIPTS (string) is the directory containing the run scripts for the current CASE.

TOOLS (string) is the directory containing CCSM utility tools.

LOGDIR (string) is the directory to which copies of the standard out log files (printout) from each of the component models will be copied.

CSMCODE (string) points to the root directory of the CCSM source code for all components.

CSMUTL (string) is the directory containing utility codes, such as the Earth System Modeling Framework (ESMF) routines.

CSMSHR (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.

CSMBLD (string) is the directory containing the makefiles and site-specific gnumake macros necessary to build the model executables.

CSMDATA (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 input data distribution tar files downloaded from the CCSM2.0 release home page (www.ccsm.ucar.edu/models/ccsm2.0).

LID (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 EXEROOT  /ptmp/$LOGNAME/$CASE  # run model here
setenv OBJROOT  $EXEROOT              # build code here
setenv LIBROOT  $EXEROOT/lib          # Location of supplemental libraries
setenv INCROOT  $LIBROOT/include      # Location of supplemental includes/modfiles
setenv ARCROOT  /XXX/$LOGNAME/archive/$CASE # archive root directory
```

EXEROOT (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.

OBJROOT (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.

LIBROOT (string) is the directory where supplemental libraries (such as ESMF) will be built and maintained.

INCROOT (string) is the directory for include and module files needed by the supplemental libraries.

ARCROOT (string) defines a directory to be used by the CCSM tools when running the data harvesting scripts.

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

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

LFSINP (string) is the local file system disk location of the input data.

LMSINP (string) is the root directory location on the local mass storage device (NCAR MSS or LANL/NERSC HPSS) of the input data.

LMSINP (string) is the root directory on the local mass storage device (NCAR MSS or LANL/NERSC HPSS) for the output data.

MACINP (string) is the remote machine to copy the inputdata from if the data cannot be located locally. The Unix scp command is used to transfer the files.

RFSINP (string) is the remote file system directory on the MACINP machine for acquiring input data via scp.

RMSINP (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.

MACOUT (string) is the remote machine to which the output data will be copied (via the Unix scp command).

RFSOUT (string) is the remote file system directory on the MACOUT machine where the output data will be sent.

```
foreach DIR ( $EXEROOT $LIBROOT $INCROOT $OBJROOT $LOGDIR)
  if !(-d $DIR) mkdir -p $DIR
end
```

This foreach loop creates the listed directories if they don't already exist.

```
#--- 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  b1.  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
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 =~ ute*   ) setenv MACH ute
 if ($MACHKEY =~ n*     ) setenv MACH nirvana
 if ($MACHKEY =~ eag*   ) setenv MACH eagle
 if ($MACHKEY =~ fal*   ) setenv MACH falcon
 setenv SITE ncar                                   # site, default is ncar
 if ($MACHKEY =~ n*     ) setenv SITE lanl
 if ($MACHKEY =~ s*     ) setenv SITE nersc
 if ($MACHKEY =~ eag*   ) setenv SITE ornl
 if ($MACHKEY =~ fal*   ) setenv SITE ornl
endif
```

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

ARCH	(string) returns the architecture of the machine on which the CCSM is being built.
SITEKEY	(string) returns the hostname of the machine on which the CCSM is being built.
SITE	(string) Converts $SITEKEY into SITE for use in the scripts. If $SITE is "unknown", then the script will halt. $SITE is used by the site-specific files $TOOLS/module.$ARCH.$SITE and $CSMBLD/Macros.$ARCH to build the CCSM.

```
echo ------------------------------------------------------------------------
echo  b2.  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  c. Select multi-processing and resolution specs
echo ------------------------------------------------------------------------

set MODELS   = (  atm   lnd   ice   ocn    cpl  )  # generic model names.
set SETUPS   = (  atm   lnd   ice   ocn    cpl  )  # setup script name

if ($GRID == T42_gx1v3 ) then
 set NTASKS  = (    8     3    16    40      1  )  # use NTASK = 1 for data model
 set NTHRDS  = (    4     4     1     1      4  )  # use NTHRD = 1 for data model
else if ($GRID == T31_gx3 ) then
 set NTASKS  = (    4     4     2     4      1  )  # use NTASK = 1 for data model
 set NTHRDS  = (    4     1     1     1      2  )  # use NTHRD = 1 for data model
else if ($GRID == T62_gx1v3 ) then
 set NTASKS  = (    1     1    16    40      1  )  # use NTASK = 1 for data model
 set NTHRDS  = (    1     1     1     1      4  )  # use NTHRD = 1 for data model
else
 echo "unsupported configuration: $GRID"
 exit 1
endif
```

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.

```
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.

```
echo ------------------------------------------------------------------------
echo  d. 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 ------------------------------------------------------------------------

#--- 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.

$SCRIPTS/$SITE.$ARCH.modules 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  d1. Build Earth System Modeling Framework   http://www.esmf.scd.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.

```
echo -------------------------------------------------------------------------
echo  d2. 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
```

In this step, the setup script is run for each CCSM component. In $EXEDIR, a log file of the build process is created. The component setup script writes its standard output into the log file. If the component setup script returns with a nonzero status, a diagnostic message is printed and the test.a1.run script exits with a nonzero return code.

```
#--- 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
```

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  e. 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_RMPOOL 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
 setenv _DSM_VERBOSE         ; setenv _DSM_PLACEMENT ROUND_ROBIN
endif
env | egrep '(MP_|LOADL|XLS|FPE|DSM|OMP|MPC)' # document above env vars
```

Settings for the IBM SP are:

limit datasize	**unlimited** maximized the virtual memory allocation.
XLSMPOPTS	**"stack=86000000"** reserves 86 Mbytes of stack space for each thread.
MP_EUILIB	**us** 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.
MP_RMPOOL	**1** tells POE to allocate nodes from resource manager pool 1.
MP_NODES	**$PROC** sets the number of nodes over which the parallel tasks will be run.
MP_PROCS	**$PROC** is the total number of processes for the model.
MP_PGMMODEL	**mpmd** identifies the programming model to be MPMD (Multiple Processes, Multiple Datastreams).
MP_CMDFILE	**poe.cmdfile** names the text file specifying the names of the component executables to be run under MPI.
MP_STDOUTMODE	**ordered** asks that standard out be buffered and flushed in the order of the tasks that wrote to standard out.
MP_SHARED_MEMORY	**yes** requests that all tasks running on the same node use shared memory for message passing on that node rather than communicating across the switch.
MP_EAGER_LIMIT	**65536** maximizes the message size of the receive data buffer for optimal performance.
MP_INFOLEVEL	**6** 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.
OMP_DYNAMIC	**FALSE** forbids the use of dynamic scheduling for OpenMP threads.
MPC_GANG	**OFF** disallows "gang scheduling" to achieve higher performance with the OMP_DYNAMIC FALSE setting.
_DSM_WAIT	**SPIN** instructs each thread to wait in a loop without giving up the CPU until a synchronization event such as a lock or barrier succeeds.
_DSM_VERBOSE	requests that all available informational messages be written to standard output
_DSM_PLACEMENT	**ROUND_ROBIN** specifies round-robin memory allocation for stack, data, and text.

```
echo ------------------------------------------------------------------------
echo  f. 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]"   &
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  g. 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 ccsm_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  h. 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  i. 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 1.4 2002/06/18 21:25:52 southern Exp $
# CVS Source:  $
# 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.ccsm.ucar.edu/models/cpl
#===============================================================================

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 ($GRID =~  T21*   ) set ATM =  (   T21  64  32 ) # name, x dimension, y dimension
if ($GRID =~  T31*   ) set ATM =  (   T31  96  48 )
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 =~  *gx3   ) set OCN =  (   gx3 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_*2*.nc

if ($GRID == T42_gx1v3) then
 set MAP_A2OF_FILE = map_T42_to_gx1v3_aave_da_010709.nc
 set MAP_A2OS_FILE = map_T42_to_gx1v3_bilin_da_010710.nc
 set MAP_O2AF_FILE = map_gx1v3_to_T42_aave_da_010709.nc
 set MAP_R2O_FILE  = map_r05_to_gx1v3_roff_smooth_010718.nc
else if ($GRID == T31_gx3) then

   [ .. many lines deleted for brevity ... ]

else if ($GRID == T85_gx1v3) then
 set MAP_A2OF_FILE = map_T85_to_gx1v3_aave_da_020405.nc
 set MAP_A2OS_FILE = map_T85_to_gx1v3_bilin_da_020405.nc
 set MAP_O2AF_FILE = map_gx1v3_to_T85_aave_da_020405.nc
 set MAP_R2O_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_A2OF_FILE $MAP_A2OF_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_A2OS_FILE $MAP_A2OS_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_O2AF_FILE $MAP_O2AF_FILE || exit 1
$TOOLS/ccsm_getinput cpl/cpl5/$MAP_R2O_FILE  $MAP_R2O_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 = $REFCASE.cpl5.r.${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'
  rest_freq   = 'monthly'
  rest_n      = 3
  diag_freq   = 'ndays'
  diag_n      = 1
  stop_option = 'ndays'
  stop_n      = 5
  hist_freq   = 'monthly'
  hist_n      = 1
  info_bcheck = 2
  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.

&inparm is the namelist group name, which matches the groupname defined within the coupler.

case_name = **'$CASE '** (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 filenaming characters such as letters, numbers, underscores, dashes, commas or periods.

case_desc = **'$CASE $CASESTR '** (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.

rest_type = **' $RUN_TYPE '** (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).

rest_date = **$BASEDATE_NUM** (string) is the initial date of the simulation. This variable is ignored on continuation or branch runs.

rest_bfile = **'${REST_BFILE}'** (string) specifies the branch file to use when starting a branch run. This ignored unless rest_type is set to 'branch'.

rest_pfile = **'$SCRIPTS/rpointer.$MODEL'** (string) is the complete filepath and filename of the restart "pointer file" used for continuation runs.

map_a2of_fn = **'$MAP_A2OF_FILE'** (string) is the filename of the map for atmosphere-to-ocean flux fields.

map_a2os_fn = **'$MAP_A2OS_FILE'** (string) is the filename of the map for atmosphere-to-ocean state fields.

map_o2af_fn = **'$MAP_O2AF_FILE'** (string) is the filename of the map for ocean-to-atmosphere flux fields.

map_r2o_fn = **'$MAP_R2O_FILE'** (string) is the filename of the map for land-runoff-to-ocean.

rest_freq = **'monthly'** (string) instructs the coupler to have all the CCSM components write out restart files on the first day of every month.

rest_n = **3** (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.

diag_freq = **'ndays'** (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.

diag_n = **1** (integer) specifies the number of time periods for the time unit set in diag_freq.

stop_option = **'ndays'** (string) controls the length of the CCSM run.

stop_n = **5** (string) specifies that this integration will run for 5 days.

hist_freq = **'monthly'** (string) controls the frequency of history file output.

hist_n = **1** (integer) is the option when hist_freq = 'ndays' or 'nstep'. Since hist_freq is 'monthly' this setting is ignored.

info_bcheck = **2** (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.

orb_year = **1990** (integer) is the calendar year that is used to determine the solar orbit and resulting solar angles.

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

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 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 VPFILE points to the Filepath list. Finally, the actual executable to be built is $EXEDIR/$MODEL.

```
# document the source code used, cleanup $EXEDIR/obj files
#-------------------------------------------------------------------

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

echo ' '
echo ================================================================================
echo  End of setup shell script  `date`
echo ================================================================================
```

The final portion of the script documents the source code CVS tags and optionally cleans up the object files that were created.

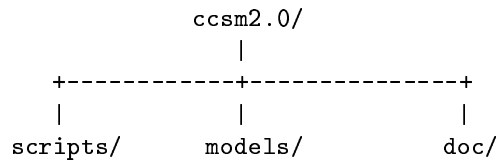
At this point, control is returned to test.a1.run.

4 Building the CCSM

CCSM2.0 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 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 CCSM on the supported platforms.

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.

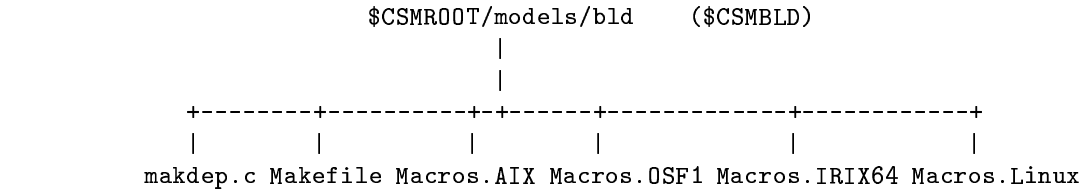
The **models/** directory contains all the source code needed to build the CCSM. Underneath the **models/** directory are directories for each of the CCSM components:

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

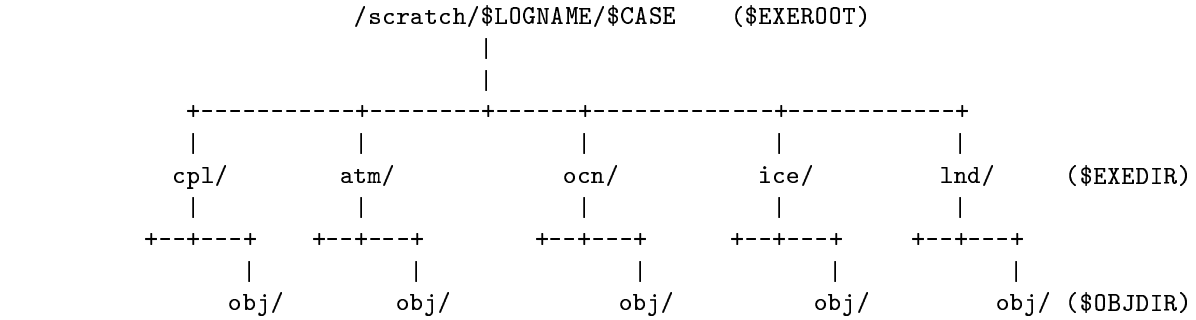


- **makdep.c** evaluates the code dependencies for all the files in all the directories in Filepath.
- **Makefile** is a generic gnumakefile for all the models.
- **Macros.AIX** contain build settings specific to AIX (IBM SP2) platforms.
- **Macros.OSF1** contain build settings specific to OSF1 (Compaq) platforms.
- **Macros.IRIX64** contain build settings specific to IRIX64 (SGI Origin) platforms.
- **Macros.Linux** contain build settings specific to Linux platforms.

The Macros files contain makefile defaults for each platform. In the current release, the Macros have been divided into different files depending on platform. Within each platform are site/machine specific options. The platform type is set in the main test script with the environment variable OS. In addition, machine dependent options are also included in the Macros files for specific platforms that have been tested. The machine specific options are set in the Macros files by use of the environment variable MACH.

4.3 $EXEDIR structure

The CCSM is built in the directory specified by $EXEROOT which is defined in the main run script, **scripts/test.a1/test.a1.run**. Underneath $EXEROOT, all of the model components will be build in their own separate subdirectories. The subdirectories will be below the $EXEDIR unless the user selects a non-defual location for $OBJDIR.



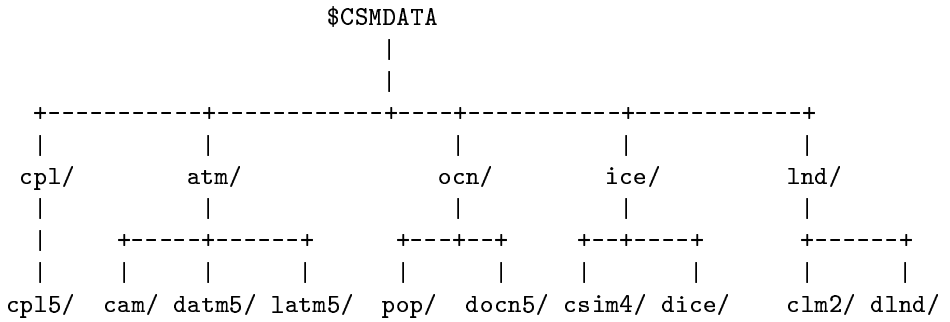
Each $EXEDIR subdirectory contains everything needed to run a specific CCSM component: the objects, executables, input and boundary datasets. Output from the component, such as standard out logs and history and restart datasets will also be written into $EXEDIR subdirectories. Some of the components, such as the ocean and sea-ice, have separate subdirectories for input, restart and history data, while the atmosphere and land put all of these in one directory.

For each component there is a directory which contains all the files created during the compilation of that component. This includes the dependency files, cpp products and object files. In this example, $OBJDIR is located below the component $EXEDIRs. However, $OBJDIR (set in test.a1.run) can be set to be completely independent of $EXEDIR if desired.

4.4 Input data positioning

The CCSM input data is distributed separately from the source code. Individual tar files exist for various resolutions and configurations. To run CCSM, these data need to be untarred from the inputdata tar file and the root location of the untarred input data needs to be used to define the variable $CSMDATA in the **$SCRIPTS/test.a1.run** script.

As with the models and $EXEDIR directories, separate directories exist for each CCSM component in the $CSMDATA directory:



$CSMDATA can be located anywhere as long as the data can be copied from that location to $EXEROOT. Once $CSMDATA has been identified in **scripts/test.a1/test.a1.run**, the individual setup scripts copy the data required for their component from $CSMDATA to the component directories under $EXEROOT.

4.5 Building the model interactively

Currently, the build and run of the CCSM model are interweaved in the same scripts. While CCSM is usually not run interactively, it is often convenient to compile the model interactively to verify initial data placement and compilation success without having to wait in a queue. There is a comment line in the main run script that when made active will cause the script to exit before CCSM begins to execute. The exit command looks like

```
#exit      # UNCOMMENT to EXIT HERE, BUILD ONLY
```

So, the CCSM model can be built interactively by making the above line active (uncommenting the line) and executing

```
./test.a1.run
```

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 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/ .

- 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/$LOGNAME/ccsm2.0/scripts
mkdir mytest1
cp test.a1/* mytest1/
cd /home/$LOGNAME/ccsm2.0/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
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 ARCROOT" to the local directory for archiving model output
```

- Run the script by submitting /home/$LOGNAME/ccsm2.0/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 my 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".

startup	represents a new case started from some model specific initial files or state.
continue	is a continuation of a case and guarantees exact restart capability.
branch	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.
hybrid	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 setup to run in batch environments and both need to be modified when using non-NCAR batch queuing 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 scr.* 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, checkin 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 CCSM2.0 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.

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.ccsm.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.

7.2 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.

8 Supporting Scripts:

A number of supporting scripts are included in the CCSM2.0 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 **j**ob **o**perating **e**nvironment (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** Echos 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 ccsm 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** [**locdir/**][**file2**] Copies a file from the local mass store to the local disk. Takes one or two arguments.
- **ccsm_mswrite** [**locdir/**]**file1 mssdir/**[**file2**] Copies a file to the local mass store from the local disk. Takes two arguments.
- **ccsm_splitdf** [**-d**]/[**-f**]/[**-h**] **dir/file** Returns the directory path or the filename for an argument like dir/file. Can handle very general cases. -d returns the directory. -f returns the file. -h is help. -f is the default return argument. Takes one argument.
- **modules.*.*** Machine-specific setup scripts to run ccsm. No arguments. Called specifically by the main ccsm 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 one month in configurations different from the default "all active models" configuration in the CCSM2.0 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 and testing 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 T42_gx1v3 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/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 for $CSMROOT, $CSMDATA and $EXEROOT. The test $CASE name is also needed.

CSMROOT (string) defines the root directory of the CCSM code distribution directory tree. The model source code, scripts and documention are located underneath this directory.

CASE (string) is the "name" that identifies this model run. The CASE name is propagated throughout the CCSM environment into script and directory names. Currently CASE can be up to 16 characters long.

CSMDATA (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 contained in the CCSM2.0 input data distribution tar files.

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.

Then, select the desired configuration. The G and M configurations are used to test the sea-ice model and require a T62_gx1v3 resolution setting from the pull-down menu. Any of the other configurations can be built at either T42_gx1v3 or T31_gx3, but these results have not yet been scientifically verified.

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. atm.setup.csh 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 1.5 2002/06/18 21:25:48 southern Exp $
# CVS $Source:
# 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.ccsm.ucar.edu/models/ccsm2
echo ===============================================================================

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

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

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

```
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 runtype setting is parsed into NSREST values appropriate for the the atmosphere model. Finally, the physics version is specified.

```
# input datasets
#-------------------------
set DATOZON = noaao3.1990.21999.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.ccsm.c011015.nc
  if ($GRID =~ T42*) set DATINIT = JAN1.T42L26.eul.ccsm.052001.nc
  if ($GRID =~ T31*) set DATINIT = SEP1.T31L26.c011005.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. While the hard-wiring of the filenames restrict the degree of automation, it ensures that the exact data 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 absortivity 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 ($GRID =~ T85*) set PARAMS = (85 256 128 26 .true. .false.  600)
if ($GRID =~ T42*) set PARAMS = (42 128 64 26 .true. .true.  1200)
if ($GRID =~ T31*) set PARAMS = (31  96 48 26 .true. .false. 1800)
if ($GRID =~ T21*) set PARAMS = (21  64 32 18 .true  .false. 2400)

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 KMXHDC =  1
if ($GRID =~ T31*) set KMXHDC =  3
if ($GRID =~ T42*  && $plev != 18) set KMXHDC =  5
if ($GRID =~ T85*  && $plev != 18) set KMXHDC =  5
```

This section defines many of the resolution dependent settings of the atmosphere model. 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 setting will be 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
 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 runtype 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 Community Atmosphere Model (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.

&camexp is the namelist group name, which matches the groupname defined within CAM.

caseid = **$CASE** (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.

ctitle = **'$CASE $CASESTR'** (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.

ncdata = **'$DATINIT'** 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.

bndtvo = **'$DATOZON'** (string) is the name of the ozone boundary dataset.

nrevsn = **'$NREVSN'** (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.

absems_data = **'$ABSDATA'** (string) is the name of the absortivity and emissivity dataset.

irt = **$MSSRPD** and **rirt** = **$MSSRPD** (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.

nswrps = **'$MSSPWD'** (string) sets the write password for any files written to the NCAR MSS.

nsrest = **$NSREST** (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).

dtime = **$DTIME** (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.

start_ymd = **$START_YMD** (integer) specifies the "basedate" for the run. This date serves as the baseline from which the current step number is calculated.

linebuf = **.false.** (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.

readtrace = **$READTRACE** (logical) specifies whether or not an input chemical tracer file is read.

nelapse = **-9999** (integer) indicates that the flux coupler will control the ending timestep for this model run.

ozncyc = **.true.** (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	**= $KMXHDC** (integer) specifies...
rest_pfile	**= '$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.

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 he 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   $plon
#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
#------------------------------------------------

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c
  gmake -j 4 -f $CSMBLD/Makefile MACFILE=$CSMBLD/Macros.$OS MODEL=cam \
      VPFILE=Filepath  EXEC=$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 VPFILE points to the Filepath list. Finally, the actual executable to be built is $EXEDIR/$MODEL.

```
echo ================================================================
echo  End of cam setup shell script
echo ================================================================
```

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:  http://www.ccsm.ucar.edu/models/ocn
#===============================================================================
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 ==  gx3) then
  setenv DT_COUNT  12
  setenv TIME_MIX_FREQ 100
  setenv INIT_TS_FILE ts_PHC2_jan_20011012

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 users guide 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 NS 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: www.ccsm.ucar.edu/models/ccsm2.0/pop/doc/POPusers_chap2.html#POPusers2.2.2

```
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 ----------------------------------------------------------------

  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 is 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#$IYEAR0#;                s#IHOUR0#0#
s#IMONTH0#$IMONTH0#;              s#IMINUTE0#0#
s#IDAY0#$IDAY0#;                  s#ISECOND0#0#

EOF

sed -f $EXEDIR/commands.sed $SRCDIR/input_templates/${OCN_GRID}_pop_in
                                          >! $EXEDIR/pop_in  || exit 3

# 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 CCSM23.0 POP User's Guide, chapter 4, Model diagnostics and output, for a complete discussion of these files (www.ccsm.ucar.edu/models/ccsm2.0/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 "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_tavg_contents** Contains a list of fields to be time-averaged.
- **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 shf.readme
 $TOOLS/ccsm_getinput $OCNDATA/forcing/sfwf_mm_all_85-88_20010320.ieeer8 sfwf

else if ($OCN_GRID == gx3) 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_20001030.ieeei4  region_mask
 $TOOLS/ccsm_getinput $OCNDATA/grid/region_mask_20001030.readme  region_mask.readme
 $TOOLS/ccsm_getinput $OCNDATA/grid/topography_20001030.ieeei4   topography
 $TOOLS/ccsm_getinput $OCNDATA/grid/topography_20001030.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 CCSM2.0. These variables are:

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

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.

```
unset echo

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

exit
```

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 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 1.5 2002/06/18 21:25:50 southern Exp $
# CVS $Source:
# 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:  http://www.ccsm.ucar.edu/models/ice-csim4
#===============================================================================

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 $CSMDATA set in test.a1.run) for the ice initial and boundary data in the CCSM2.0 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 ($PRESCRIBED_ICE == .true.) then
  if ($PRESCRIBED_ICE_CLIM == .false. && $BASEDATE != 1977-01-01) then
    echo "Set BASEDATE in main script to 1977-01-01" ; exit -1
  endif
  if ($RUNTYPE == startup) set RESTART = .false.
  if ($ICE_GRID == 'gx3') \
      set PICE_DATA = AMIP_bc${ICE_GRID}_1976-1996_011030.nc
  if ($ICE_GRID == 'gx1v3') \
      set PICE_DATA = AMIP_bc${ICE_GRID}_1976-1996_010817.nc
  $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
```

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
  $TOOLS/ccsm_getinput $ICEDATA/pop_frc_${ICE_GRID}_010815.nc
                                        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
```

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 OCEAN-MIXED_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 $CASESTR'
  , runtype      = '$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.
/
```

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.

&ice_nml	is the namelist group name, which matches the groupname defined within the sea-ice model.
runid	**= '$CASE $CASESTR'** (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 $SCRIPTS/test.a1.run script.
runtype	**= '$RUNTYPE'** (string) specifies the state in which this run is to begin. See the sample setup script section for discussion of this variable.
istep0	**= 0** (integer) is the starting step number used to set the current elapsed time of the run on startup.
dt	**= 3600.0** (real) sets the timestep for the ice transport and thermodynamics to 3600 seconds.
ndte	**= 120** (integer) sets the number of ice model dynamics subcycles per model timestep to damp elastic waves.
npt	**= 99999** (integer) sets the total number of timesteps. This is ignored in a coupled run where the flux coupler determines the stop point.
diagfreq	**= 24** (integer) results in diagnostic printout being output at noon every 24 hours.
histfreq	**= 'm'** (string) requests that history output be written monthly.
dumpfreq	**= 'y'** (string) requests that restart output be written yearly. In a coupled run, this variable is ignored and instructions from the coupler are used.
hist_avg	**= .true.** (logical) defines the output history data as being averaged over the output period.
restart	**= $RESTART** (logical) flags whether or not to read the restart data filenames from the restart pointer files.
print_points	**= .false.** (logical) flags whether or not to print out grid point data.
kitd	**= 1** (integer) determines the type of itd conversions (0 = delta f, 1 = linear).
kdyn	**= 1** (integer) determines the type of ice dynamics. Currently, only elastic-viscous-plastic dynamics (kdyn=1) is supported.
kstrength	**= 1** (integer) instructs the ice to use Rothrock 1975 pressure formulation.
evp_damping	**= .false.** (logical) turns off the elastic-viscous-plastic ice damping.
snow_into_ocn	**= .false.** (logical) keeps ridging snow from being dumped into the ocean.
advection	**= 'mpdata2'** (string) set the ice transport advection algorithm to a second order advection scheme using mpdata,
grid_type	**= 'displaced_pole'** (string) sets the ice grid to displace_pole rather than rectangular (default).
grid_file	**= 'data.domain.grid'** (string) names the file containing the ice grid information.
kmt_file	**= 'data.domain.kmt'** (string) names the file containing the land-mask information.
incond_dir	**= '$INIDIR/'** (string) is the directory for snapshot initial conditions.
incond_file	**= '$CASE.csim.i.'** (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.
/ marks the end of the ice_nml group of namelist variables.
&icefields_nml 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 build 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 ( $GRID =~ *gx1v3) set RES = 320x384
if !($?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/.\*x//`
```

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 = $NLON
setenv NY 1
if ($NTASK > 16) setenv NY 2
@ 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
#-------------------------------------------------------------------------

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c                              || exit 2
  if ($NTHRD > 1) setenv THREAD TRUE
  gmake -j 6 VPFILE=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 VPFILE 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. 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 1.6 2002/06/18 21:25:51 southern Exp $
# CVS $Source:
# 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:  http://www.cgd.ucar.edu/cms/lsm/clm/clm.html
#===============================================================================

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 $DATINIT) exit 99
 else
  set DATINIT = ""
 endif
else if ($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}_${atm_setup}.${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.

&clmexp	is the namelist group name, which matches the groupname defined within the land-surface model.
rtm_nsteps	= **$RTM_NSTEPS** (integer) sets the number of timesteps between river runoff calculations.
caseid	= **'$CASE'** (string) sets a unique, 16-character, test string used to identify this run. The CASE variable is set in $SCRIPTS/test.a1.run script and is used extensively in the CLM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX file-naming characters such as letters, numbers, underscores, dashes, commas or periods.
ctitle	= **'$CASE $CASESTR'** (string) provides 80 characters to further describe the run. This description appears in the output logs and in the header data for the output data sets. CASESTR is set in the run script.
nsrest	= **$NSREST** (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).
start_ymd	= **$BASEDATE_NUM** (integer) specifies the "basedate" for the run. This date serves as the baseline from which the current step number is calculated.
start_tod	= **0** (integer) sets the starting time of day for the run in seconds.
nelapse	= **-9999** (integer) indicates that the flux coupler will control the ending timestep for this model run.
dtime	= **$DTIME** (integer) sets the timestep, in seconds, for the land model. The timestep is dependent on the resolution of the model.
irad	= **-1** (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.
csm_doflxave	= **.true.** (logical) instructs the land model to only communicate with flux coupler on albedo calculation timesteps.
nrevsn	= **'$NREVSN '** (string) is the optional name of the branch restart dataset to be read if this is a branch run. This dataset is a binary file containing the exact state of a previous run.
finidat	= **'$DATINIT '** (string) sets the name of the initial data file. If $DATINIT is blank, then the land model is initialized from a "cold start" condition.
fsurdat	= **'$surf_dat'** (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.
fpftcon	= **'pft-physiology-vegdyn-cleanup-ratio'** (string) identifies the Plant Functional Type (PFT) physiological constants input dataset.
frivinp_rtm	= **'rdirc.05'** (string) is the name of the River Transport Model (RTM) input dataset.
mksrf_fvegtyp	= **'mksrf_pft.nc'** (string) is the vegetation type data file name.
mksrf_fsoitex	= **'mksrf_soitex.10level.nc'** (string) is the soil texture data file name.
mksrf_fsoicol	= **'mksrf_soicol_clm2.nc'** (string) is the soil color data file name.
mksrf_flanwat	= **'mksrf_lanwat.nc'** (string) is the inland water data file name.
mksrf_fglacier	= **'mksrf_glacier.nc'** (string) is the glacier data file name.
mksrf_furban	= **'mksrf_urban.nc'** (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 aveerage)
hist_crtinic	**= 'YEARLY'** (string) outputs a land initial condition file if set to 'MONTHLY' or 'YEARLY',
mss_irt	**= $MSSRPD** (integer) specifies the mass store retention period for output history and restart files when running at NCAR. If irt is set to 0, the output history and restart files are not written to the Mass Storage System.
rpntpath	**= '$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/mksrfdata
$CAMLND_SHARE
$CSMSHR
$CSMCODE/utils/timing
EOF
```

The land-surface model is build 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 land model document www.ccsm.ucar.edu/models/ccsm2.0/UsersGuide/UsersGuide). for information on running the land model as a "standalone" model.

```
# Create the executable
#----------------------
if ($BLDTYPE == '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 "BLDTYPE = $BLDTYPE"
endif
wait
```

The land model executable is only built if $BLDTYPE 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.

```
echo ============================================================
echo  End of clm2 model setup shell script
echo ============================================================
```

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 CCSM2.0 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. The test data referenced in these scripts uses CSM1 forcing data and should not be expected to return realistic results for the CCSM2.0 model.

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:
# CVS $Source:
# CVS $Name:
#===============================================================================
# datm.setup.csh: Prepare a CSM data atmosphere component, datm5, for execution
#                 For help, see:  http://www.ccsm.ucar.edu/models/atm-datm
#===============================================================================

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 =~ T42*) set DATA_DIR = mss:/BOVILLE/csm/f015.00/datm/data
if ( $GRID =~ T42*) set DATA_DIR = cp:$CSMDATA/atm/datm5
if ( $GRID =~ T31*) set DATA_DIR = mss:/SHIELDS/csm/f023.02/datm/data

#rm -f 0005-01.nc
#rm -f 0009-12.nc
#if ( $GRID =~ T42*) $TOOLS/ccsm_getinput atm/datm5/0005-01.nc
#if ( $GRID =~ T42*) $TOOLS/ccsm_getinput atm/datm5/0009-12.nc

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_year0 =  5
  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.

&inparm	is the namelist group name, which matches the groupname defined within the data component.
caseid	= **$CASE** (string) is a unique string (16 characters or less) that is used to identify this run. The CASE variable is set in $SCRIPTS/test.a1.run and is used extensively in the CCSM as an identifier. Since CASE will be used in file and directory names, it should only contain standard UNIX file-naming characters such as letters, numbers, underscores, dashes, commas or periods.
ctitle	= **'$CASE $CASESTR'** (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.
rest_type	= **' $RUN_TYPE '** (string) specifies the state in which the run is to be started. rest_type settings initial, branch and continue map into the CCSM variables (startup/hybrid branch and continue).
ncpl	= **24** (integer) sets the coupling frequency to 24 times per day.
data_dir	= **'$DATA_DIR'** (string) supplies the location of the input data directory to the code.
data_year0	= **5** (integer) sets the first year of the input data to be year 5.
data_nyear	= **1** (integer) sets the total number of years to cycle the input data over to 1.
data_oyear	= **1** (integer) defines the coupler year corresponding to data_year0. This allows data with arbitrary dates to be input.
flux_albfb	= **1** (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 $EXEDIR/obj
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.

cmp -s $CSMCODE/atm/datm5/dims.h.$ATM_GRID dims.h || \
    cp $CSMCODE/atm/datm5/dims.h.$ATM_GRID dims.h  || exit 3
```

The data component executable is build 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
#-------------------------------------------------------------------

if ($BLDTYPE == 'true') then
  cc -o makdep $CSMBLD/makdep.c                                || exit 2
  if ($NTHRD > 1) setenv THREAD TRUE
  gmake -j 6 VPFILE=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 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 VPFILE 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 ================================================================================
```

The final portion of the script documents the CVS tags of the source code and has an optional, commented out, line to clean up the object files that were created.

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 CCSM2.0 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.