CESM User's Guide (CESM1.2 Release Series User's Guide) (PDF[1])

CESM Software Engineering Group (CSEG)
NCAR

CESM User's Guide (CESM1.2 Release Series User's Guide) (PDF[1])
by CESM Software Engineering Group (CSEG)

Table of Contents

Chapter 1. Introduction

How To Use This Document

This guide instructs both novice and experienced users on building and running CESM. If you are a new user, we recommend that the introductory sections be read before moving onto other sections or the Quick Start procedure. This document is written so that, as much as possible, individual sections stand on their own and the user's guide can be scanned and sections read in a relatively ad hoc order. In addition, the web version provides clickable links that tie different sections together.

The chapters attempt to provide relatively detailed information about specific aspects of CESM such as setting up a case, building the model, running the model, porting, and testing. There is also a large section of use cases and FAQs.

```
Throughout the document, this presentation style indicates shell
commands and options, fragments of code, namelist variables, etc.
Where examples from an interactive shell session are presented,
lines starting with '>' indicate the shell prompt.
```

Please feel free to provide feedback to CESM about how to improve the documentation.

CESM Model Version Naming Conventions

CESM model release versions include three numbers separated by a dot (.) - CESM X.Y.Z

- X - corresponds to the major release number indicating significant science changes.
- Y - corresponds to the addition of new infrastructure and new science capabilities for targeted components.
- Z - corresponds to release bug fixes and machine updates.

CESM Overview

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

The CESM system can be configured a number of different ways from both a science and technical perspective. CESM supports numerous resolutions[1], and component configurations[2]. In addition, each model component has input options to configure specific model physics and parameterizations. CESM can be run on a number of different hardware platforms[3], and has a relatively flexible design with respect to processor layout of components. CESM also supports both an internally developed set of component interfaces and the ESMF compliant component interfaces (See the Section called *BASICS: How do I use the ESMF library and ESMF interfaces?* in Chapter 6)

The CESM project is a cooperative effort among U.S. climate researchers. Primarily supported by the National Science Foundation (NSF) and centered at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, the CESM project enjoys close collaborations with the U.S. Department of Energy and the National Aeronautics and Space Administration. Scientific development of the CESM is guided by the CESM working groups, which meet twice a year. The main CESM

workshop is held each year in June to showcase results from the various working groups and coordinate future CESM developments among the working groups. The CESM website[4] provides more information on the CESM project, such as the management structure, the scientific working groups, downloadable source code, and online archives of data from previous CESM experiments.

CESM Software/Operating System Prerequisites

The following are the external system and software requirements for installing and running CESM.

- UNIX style operating system such as CNL, AIX and Linux
- csh, sh, and perl scripting languages
- subversion client version 1.4.2 or greater
- Fortran (2003 recommended, 90 required) and C compilers. pgi, intel, and xlf are recommended compilers.
- MPI (although CESM does not absolutely require it for running on one processor)
- NetCDF 4.2.0 or newer[5].
- ESMF 5.2.0 or newer (optional)[6].
- pnetcdf 1.2.0 is required and 1.3.1 is recommended[7]
- Trilinos[8] may be required for certain configurations
- LAPACKm[9] or a vendor supplied equivalent may also be required for some configurations.
- CMake 2.8.6 or newer[10] is required for configurations that include CISM.

The following table contains the version in use at the time of release. These versions are known to work at the time of the release for the specified hardware.

Table 1-1. Recommmended Software Package Versions by Machine

Machine	Version Recommendations
Cray XT Series	pgf95 12.4.0
IBM Power Series	xlf 12.1, xlC 10.1
IBM Bluegene/P	xlf 12.01, xlC 10.01
Linux Machine	ifort, icc (intel64) 12.1.4

Caution

NetCDF must be built with the same Fortran compiler as CESM. In the netCDF build the FC environment variable specifies which Fortran compiler to use. CESM is written mostly in Fortran, netCDF is written in C. Because there is no standard way to call a C program from a Fortran program, the Fortran to C layer between CESM and netCDF will vary depending on which Fortran compiler you use for CESM. When a function in the netCDF library is called from a Fortran application, the netCDF Fortran API calls the netCDF C library. If you do not use the same compiler to build netCDF and CESM you will in most cases get errors from netCDF saying certain netCDF functions cannot be found.

Parallel-netCDF, also referred to as pnetcdf, is optional. If a user chooses to use pnetcdf, version 1.2.0 or later should be used with CESM. It is a library that is file-format compatible with netCDF, and provides higher performance by using MPI-IO.

Pnetcdf is enabled by setting the PNETCDF_PATH variable in the Macros file. You must also specify that you want pnetcdf at runtime via the io_typename argument that can be set to either "netcdf" or "pnetcdf" for each component.

CESM Components

CESM consists of seven geophysical models: atmosphere (atm), sea-ice (ice), land (lnd), river-runoff (rof), ocean (ocn), land-ice (glc), and ocean-wave (wav - stub only), plus a coupler (cpl) that coordinates the geophysics models time evolution and passes information between them. Each model may have "active," "data," "dead," or "stub" component version allowing for a variety of "plug and play" combinations.

During the course of a CESM run, the model components integrate forward in time, periodically stopping to exchange information with the coupler. The coupler meanwhile receives fields from the component models, computes, maps, and merges this information, then sends the fields back to the component models. The coupler brokers this sequence of communication interchanges and manages the overall time progression of the coupled system. A CESM component set is comprised of seven components: one component from each model (atm, lnd, rof, ocn, ice, glc, and wav) plus the coupler. Model components are written primarily in Fortran 90/95/2003.

The active (dynamical) components are generally fully prognostic, and they are state-of-the-art climate prediction and analysis tools. Because the active models are relatively expensive to run, data models that cycle input data are included for testing, spin-up, and model parameterization development. The dead components generate scientifically invalid data and exist only to support technical system testing. The dead components must all be run together and should never be combined with any active or data versions of models. Stub components exist only to satisfy interface requirements when the component is not needed for the model configuration (e.g., the active land component forced with atmospheric data does not need ice, ocn, or glc components, so ice, ocn, and glc stubs are used).

The CESM components can be summarized as follows:

Model Type	Model Name	Component Name	Type	Description
atmosphere	atm	cam	active	The Community Atmosphere Model (CAM) is a global atmospheric general circulation model developed from the NCAR CCM3.
atmosphere	atm	datm	data	The data atmosphere component is a pure data component that reads in atmospheric forcing data
atmosphere	atm	xatm	dead	

Model Type	Model Name	Component Name	Type	Description
atmosphere	atm	satm	stub	
land	lnd	clm	active	The Community Land Model (CLM) is the result of a collaborative project between scientists in the Terrestrial Sciences Section of the Climate and Global Dynamics Division (CGD) at NCAR and the CESM Land Model Working Group. Other principal working groups that also contribute to the CLM are Biogeochemistry, Paleoclimate, and Climate Change and Assessment.
land	lnd	dlnd	data	The data land component no longer has data runoff functionality. It works as a purely data-land component (reading in coupler history data for atm/land fluxes and land albedos produced by a previous run) or both.
land	lnd	xlnd	dead	
land	lnd	slnd	stub	

Model Type	Model Name	Component Name	Type	Description
river-runoff	rof	rtm	active	The river transport model (RTM) was previously part of CLM and was developed to route total runoff from the land surface model to either the active ocean or marginal seas which enables the hydrologic cycle to be closed (Branstetter 2001, Branstetter and Famiglietti 1999). This is needed to model ocean convection and circulation, which is affected by freshwater input.
river-runoff	rof	drof	data	The data runoff model was previously part of the data land model and functions as a purely data-runoff model (reading in runoff data).
river-runoff	rof	xrof	dead	
river-runoff	rof	srof	stub	
ocean	ocn	pop	active	The ocean model is an extension of the Parallel Ocean Program (POP) Version 2 from Los Alamos National Laboratory (LANL).

Model Type	Model Name	Component Name	Type	Description
ocean	ocn	docn	data	The data ocean component has two distinct modes of operation. It can run as a pure data model, reading ocean SSTs (normally climatological) from input datasets, interpolating in space and time, and then passing these to the coupler. Alternatively, docn can compute updated SSTs based on a slab ocean model where bottom ocean heat flux convergence and boundary layer depths are read in and used with the atmo-sphere/ocean and ice/ocean fluxes obtained from the coupler.
ocean	ocn	xocn	dead	
ocean	ocn	socn	stub	

Model Type	Model Name	Component Name	Type	Description
sea-ice	ice	cice	active	The sea-ice component (CICE) is an extension of the Los Alamos National Laboratory (LANL) sea-ice model and was developed though collaboration within the CESM Polar Climate Working Group (PCWG). In CESM, CICE can run as a fully prognostic component or in prescribed mode where ice coverage (normally climatological) is read in.
sea-ice	ice	dice	data	The data ice component is a partially prognostic model. The model reads in ice coverage and receives atmospheric forcing from the coupler, and then it calculates the ice/atmosphere and ice/ocean fluxes. The data ice component acts very similarly to CICE running in prescribed mode.
sea-ice	ice	xice	dead	
sea-ice	ice	sice	stub	

Model Type	Model Name	Component Name	Type	Description
land-ice	glc	cism	active	The CISM component is an extension of the Glimmer ice sheet model.
land-ice	glc	sglc	stub	
ocean-wave	wav	xwav	dead	Support for a separate ocean wave component has been added to the system. At the present time, only stub and dead versions of the wave model are available in this release. Development of a prognostic wave model is underway, and it may be added to the system at some future time.
ocean-wave	wav	swav	stub	
coupler	cpl	cpl	active	The CESM coupler was built primarily through a collaboration of the NCAR CESM Software Engineering Group and the Argonne National Laboratory (ANL). The MCT coupling library provides much of the infrastructure.

CESM Component Sets

The CESM components can be combined in numerous ways to carry out various scientific or software experiments. A particular mix of components, *along with* component-specific configuration and/or namelist settings is called a component

set or "compset." CESM has a shorthand naming convention for component sets that are supported out-of-the-box.

The compset name usually has a well defined first letter followed by some characters that are indicative of the configuration setup. Each compset name has a corresponding short name. Users are not limited to the predefined component set combinations. A user may define their own component set.

See supported component sets[11] for a complete list of supported compset options. Running **create_newcase** with the option "-list" will also always provide a listing of the supported out-of-the-box component sets for the local version of CESM.

In general, the first letter of a compset name indicates which components are used. An exception to this rule is the use of "G" as a second letter to indicate use of the active glc model, CISM.

CESM Grids

The grids are specified in CESM by setting an overall model resolution. Once the overall model resolution is set, components will read in appropriate grids files and the coupler will read in appropriate mapping weights files. Coupler mapping weights are always generated externally in CESM. The components will send the grid data to the coupler at initialization, and the coupler will check that the component grids are consistent with each other and with the mapping weights files.

In CESM1.2, the ocean and ice must be on the same grid, but the atmosphere and land and river runoff can each be on different grids. Each component determines its own unique grid decomposition based upon the total number of pes assigned to that component.

CESM supports several types of grids out-of-the-box including single point, finite volume, spectral, cubed sphere, displaced pole, and tripole. This page, Conservative Remapping on Spherical Grids[12], illustrates a number of these grid types. These grids are used internally by the models. Input datasets are usually on the same grid but in some cases, they can be interpolated from regular lon/lat grids in the data models. The finite volume and spectral grids are generally associated with atmosphere and land models but the data ocean and data ice models are also supported on those grids. The cubed sphere grid is used only by the active atmosphere model, cam. And the displaced pole and tripole grids are used by the ocean and ice models. Not every grid can be run by every component. The ocean and ice models run on either a Greenland dipole or a tripole grid. The Greenland Pole grid[13] is a latitude/longitude grid, with the North Pole displaced over Greenland to avoid singularity problems in the ocn and ice models. The low-resolution Greenland pole mesh from CCSM3 is illustrated in Yeager et al., "The Low-Resolution CCSM3", AMS (2006), Figure 1b., Web.[14] Similarly, the Poseidon tripole grid [15] is a latitude/longitude grid with three poles that are all centered over land.

CESM1.2 has a completely new naming convention for model resolutions . Using this naming convention, the complete list of currently supported grid resolutions can be viewed at supported resolutions page[16].

CESM Machines

Scripts for supported machines and userdefined machines are provided with the CESM release. Supported machines have machine specific files and settings added to the CESM scripts and are machines that should run CESM cases out-of-the-box. Machines are supported in CESM on an individual basis and are usually listed by their common site-specific name. To get a machine ported and functionally supported in CESM, local batch, run, environment, and compiler information must be configured in the CESM scripts. The machine name "userdefined" machines refer to any machine that the user defines and requires that a user edit the resulting xml files

to fill in information required for the target platform. This functionality is handy in accelerating the porting process and quickly getting a case running on a new platform. For more information on porting, see Chapter 5. The list of available machines are documented in CESM supported machines[17]. Running create_newcase with the "-list" option will also show the list of available machines for the current local version of CESM. Supported machines have undergone the full CESM porting process. The machines available in each of these categories changes as access to machines change over time.

CESM Validation

Although CESM can be run out-of-the-box for a variety of resolutions, component combinations, and machines, MOST combinations of component sets, resolutions, and machines have not undergone rigorous scientific climate validation. Control runs accompany "scientifically supported" component sets and resolution and are documented on the release page. These control runs should be scientifically reproducible on the original platform or other platforms. Bit-for-bit reproducibility cannot be guaranteed due to variations in compiler or system versions. Users should carry out their own validations on any platform prior to doing scientific runs or scientific analysis and documentation.

Downloading CESM

Downloading the code and scripts - starting with CESM1.2.1

*** IMPORTANT NOTE ** Starting with CESM1.2.1, the Subversion repository path has changed. All documetation for downloading the most current version of the model has been updated to reflect this change and older version differences are noted.*

CESM release code will be made available through a Subversion repository. Access to the code will require Subversion client software in place that is compatible with our Subversion server software, such as a recent version of the command line client, svn. Currently, our server software is at version 1.7.4. We recommend using a client at version 1.6 or later, though older versions may suffice. We cannot guarantee a client older than 1.4.2. For more information or to download open source tools, visit:

`http://subversion.tigris.org/`[18]

With a valid svn client installed on the machine where CESM will be built and run, the user may download the latest version of the release code. First view the available release versions with one of the following commands:

** IMPORTANT NOTE ** Starting with CESM1.2.1, the Subversion repository path has changed.

```
> svn list https://svn-ccsm-models.cgd.ucar.edu/cesm1/release_tags
```

For all versions prior to CESM1.2.1, please use the following command to view available releases.

```
> svn list https://svn-ccsm-release.cgd.ucar.edu/model_versions
```

When contacting the Subversion server for the first time, the following certificate message will likely be generated:

```
Error validating server certificate for
'https://svn-ccsm-models.cgd.ucar.edu:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
 - Hostname: *.cgd.ucar.edu
 - Valid: from Tue, 12 Jun 2012 00:00:00 GMT until Wed, 17 Jun 2015
12:00:00 GMT
 - Issuer: www.digicert.com, DigiCert Inc, US
 - Fingerprint: eb:30:7d:c5:06:e6:b1:6f:e8:4f:c6:0a:79:6f:af:ec:5c:18:e2:32
(R)eject, accept (t)emporarily or accept (p)ermanently? p
```

After accepting the certificate, the following authentication message will likely be generated:

```
svn list https://svn-ccsm-models.cgd.ucar.edu/cesm1/release_tags
Authentication realm: <https://svn-ccsm-models.cgd.ucar.edu:443> ccsm:models
Password for '[username]':
Authentication realm: <https://svn-ccsm-models.cgd.ucar.edu:443D> ccsm:models
Username: guestuser
Password for 'guestuser':

------------------------------------------------------------------------
ATTENTION!  Your password for authentication realm:

   <https://svn-ccsm-models.cgd.ucar.edu:443> ccsm:models

can only be stored to disk unencrypted!  You are advised to configure
your system so that Subversion can store passwords encrypted, if
possible.  See the documentation for details.

You can avoid future appearances of this warning by setting the value
of the 'store-plaintext-passwords' option to either 'yes' or 'no' in
'/glade/u/home/[username]/.subversion/servers'.
------------------------------------------------------------------------
Store password unencrypted (yes/no)? yes
cesm1_2_1/
```

Be aware that the request is set to the current machine login id and you must enter the CESM registered default username of 'guestuser' by pressing the 'Enter' key when prompted for a Username.

You may be prompted up to 3 times for the username and password when checking out the code for the first time from this new Subversion path. This is because the code is distributed across a number of different Subversion repositories and each repository requires authentication.

Once correctly entered, the username and password will be cached in a protected subdirectory of the user's home directory so that repeated entry of this information will not be required for a given machine.

The release tags should follow a recognizable naming pattern, and they can be checked out from the central source repository into a local sandbox directory. The following example shows how to checkout model version CESM1.2.1:

```
> svn co https://svn-ccsm-models.cgd.ucar.edu/cesm1/release_tags/cesm1_2_1 cesm1_2_1
```

Caution

If a problem was encountered during checkout, which may happen with an older version of the client software, it may appear to have downloaded successfully, but in fact only a partial checkout has occurred. To ensure a successful download, make sure the last line of svn output has the following statement:

```
Checked out revision XXXXX.
```

*** IMPORTANT NOTE ** Starting with CESM1.2.1, the Subversion repository path has changed. Consequently, the information below regarding 'svn update' or 'svn switch' is only valid for CESM releases prior to CESM1.2.1.*

Or, in the case of an 'svn update' or 'svn switch':

```
Updated to revision XXXXX.
```

This will create a directory called `cesm1_2_1` that can be used to modify, build, and run the model. The following Subversion subcommands can be executed in the working sandbox directory.

For various information regarding the release version checked out...

```
> svn info
```

For a listing of files that have changed since checkout...

```
> svn status
```

For a description of the changes made to the working copy...

```
> svn diff
```

Obtaining new release versions of CESM - prior to CESM1.2.1

*** IMPORTANT NOTE ** Starting with CESM1.2.1, the Subversion repository path has changed. Consequently, the information below regarding 'svn update' or 'svn switch' is only valid for CESM releases prior to CESM1.2.1. Follow the steps outlined above to upgrade to CESM1.2.1 from previous versions of the model.*

To update to a newer version of the release code you can download a new version of CESM from the svn central source repository in the following way:

Suppose for example that a new version of CESM is available at https://svn-ccsm-release.cgd.ucar.edu/model_versions/cesm1_<X_newversion>. This version can be checked out directly using the same standard CESM download method.

As an alternative, some users may find the svn switch operation useful. In particular, if you've used svn to check out the previous release, cesm1_<X_previousversion>, and if you've made modifications to that code, you should consider using the svn switch operation. This operation will not only upgrade your code to the version cesm1_<X_newversion>, but will also attempt to reapply your modifications to the newer version.

How to use the svn switch operation:

Suppose you've used svn to check out cesm1_<X_previousversion> into the directory called /home/user/cesm1_1_1

1. Make a backup copy of /home/user/cesm1_1_1 -- this is important in case you encounter any problems with the update
2. cd to the top level of your cesm1_1 code tree...

```
> cd /home/user/cesm1_1_1
```

3. Issue the following svn command...

```
> svn switch https://svn-ccsm-release.cgd.ucar.edu/model_versions/cesm1_<X_newver
```

The svn switch operation will upgrade all the code to the new cesm1_<X_newversion> version, and for any files that have been modified, will attempt to reapply those modifications to the newer code.

Note that an update to a newer version of the release code may result in conflicts with modified files in the local working copy. These conflicts will likely require that differences be resolved manually before use of the working copy may continue. For help in resolving svn conflicts, please visit the subversion website,

http://svnbook.red-bean.com/en/1.5/svn.tour.cycle.html#svn.tour.cycle.resolve[19]

A read-only option is available for users to view via a web browser at

https://svn-ccsm-release.cgd.ucar.edu

where the entire CESM release directory tree can be navigated.

The following examples show common problems and their solutions.

Problem 1: If the hostname is typed incorrectly:

```
> svn list https://svn-ccsm-release.cgd.ucar.edu/model_versions/cesm1_1_<version>
svn: PROPFIND request failed on '/model_versions/cesm1_1_<version>'
svn: PROPFIND of '/model_versions/cesm1_1_<version>': Could not resolve hostname `svn-c
```

Problem 2: If http is typed instead of https:

```
> svn list http://svn-ccsm-release.cgd.ucar.edu/model_versions/cesm1_1_<version>
svn: PROPFIND request failed on '/model_versions/cesm1_1_<version>'
svn: PROPFIND of '/model_versions/cesm1_1_<version>': could not connect to server (http
```

Downloading input data

Input datasets are needed to run the model. CESM input data will be made available through a separate Subversion input data repository. The username and password for the input data repository will be the same as for the code repository.

> **Note:** The input data repository contains datasets for many configurations and resolutions and is well over 1 TByte in total size. DO NOT try to download the entire dataset.

Datasets can be downloaded on a case by case basis as needed and CESM now provides tools to check and download input data automatically.

A local input data directory should exist on the local disk, and it also needs to be set in the CESM scripts via the variable $DIN_LOC_ROOT. For supported machines, this variable is preset. For generic machines, this variable is set as an argument to **create_newcase**. Multiple users can share the same $DIN_LOC_ROOT directory.

The files in the subdirectories of $DIN_LOC_ROOT should be write-protected. This prevents these files from being accidentally modified or deleted. The directories in $DIN_LOC_ROOT should generally be group writable, so the directory can be shared among multiple users.

As part of the process of generating the CESM executable, the utility, **check_input_data** is called, and it attempts to locate all required input data for the case based upon file lists generated by components. If the required data is not found on local disk in $DIN_LOC_ROOT, then the data will be downloaded automatically by the scripts or it can be downloaded by the user by invoking **check_input_data** with the -export command argument. If you want to download the input data manually you should do it before you build CESM.

It is possible for users to download the data using svn subcommands directly, but use of the **check_input_data** script is highly recommended to ensure that only the required datasets are downloaded. Again, users are STRONGLY DISCOURAGED from downloading the entire input dataset from the repository due to the size.

Quick Start (CESM Workflow)

The following quick start guide is for versions of CESM that have already been ported to the local target machine. If CESM has not yet been ported to the target machine, please see Chapter 5. If you are new to CESM, please consider reading the introduction first

These definitions are required to understand this section:

- $COMPSET refers to the component set.
- $RES refers to the model resolution.
- $MACH refers to the target machine.
- $CCSMROOT refers to the CESM root directory.
- $CASE refers to the case name.
- $CASEROOT refers to the full pathname of the root directory where the case ($CASE) will be created.
- $EXEROOT refers to the executable directory. ($EXEROOT is normally NOT the same as $CASEROOT).
- $RUNDIR refers to the directory where CESM actually runs. This is normally set to $EXEROOT/run. (Note: changing $EXEROOT does not change $RUNDIR as these are independent variables.)

This is the procedure for quickly setting up and running a CESM case.

1. Download CESM (see Download CESM).
2. Select a machine, a component set, and a resolution from the list displayed after invoking this command:

   ```
   > cd $CCSMROOT/scripts
   > create_newcase -list
   ```

   See the supported component sets[21], supported model resolutions[22] and supported machines[23]. for a complete list of CESM supported component sets, grids and computational platforms.
3. Create a case.

   The **create_newcase** command creates a case directory containing the scripts and xml files to configure a case (see below) for the requested resolution, component set, and machine. **create_newcase** has several required arguments and if a generic machine is used, several additional options must be set (invoke create_newcase -h for help).

   If running on a supported machine, ($MACH), then invoke **create_newcase** as follows:

```
> create_newcase -case $CASEROOT \
         -mach $MACH \
         -compset $COMPSET \
         -res $RES
```

If running on a new target machine, see porting in Chapter 5.

4. Setting up the case run script

   Issuing the **cesm_setup** command creates a $CASEROOT/$CASE.run script along with user_nl_xxx files, where xxx denotes the set of components for the given case configuration. Before invoking **cesm_setup**, modify the env_mach_pes.xml file in $CASEROOT as needed for the experiment.

   a. **cd** to the $CASEROOT directory.

      ```
      > cd $CASEROOT
      ```

   b. Modify settings in `env_mach_pes.xml` (optional). (Note: To edit any of the env xml files, use the xmlchange command. invoke **xmlchange** -h for help.)

   c. Invoke the **cesm_setup** command.

      ```
      > ./cesm_setup
      ```

5. Build the executable.

   a. Modify build settings in `env_build.xml` (optional).

   b. Run the build script.

      ```
      > $CASE.build
      ```

6. Run the case.

   a. Modify runtime settings in `env_run.xml` (optional). In particular, set the $DOUT_S variable to FALSE.

   b. Submit the job to the batch queue.

      ```
      > $CASE.submit
      ```

7. When the job is complete, review the following directories and files

   a. $RUNDIR. This directory is set in the env_build.xml file. This is the location where CESM was run. There should be log files there for every component (ie. of the form cpl.log.yymmdd-hhmmss). Each component writes its own log file. Also see whether any restart or history files were written. To check that a run completed successfully, check the last several lines of the cpl.log file for the string " SUCCESSFUL TERMINATION OF CPL7-CCSM ".

   b. $CASEROOT/logs. The log files should have been copied into this directory if the run completed successfully.

   c. $CASEROOT. There could be a standard out and/or standard error file.

   d. $CASEROOT/CaseDocs. The case namelist files are copied into this directory from the $RUNDIR.

   e. $CASEROOT/timing. There should be a couple of timing files there that summarize the model performance.

   f. $DOUT_S_ROOT/$CASE. This is the archive directory. If $DOUT_S is FALSE, then no archive directory should exist. If $DOUT_S is TRUE, then log, history, and restart files should have been copied into a directory tree here.

Notes

1. ../modelnl/grid.html
2. ../modelnl/compsets.html
3. ../modelnl/machines.html
4. http://www2.cesm.ucar.edu/
5. http://www.unidata.ucar.edu/software/netcdf/
6. http://www.earthsystemmodeling.org/
7. http://trac.mcs.anl.gov/projects/parallel-netcdf/
8. http://trilinos.sandia.gov/
9. http://www.netlib.org/lapack/
10. http://www.cmake.org/
11. ../modelnl/compsets.html
12. http://www.image.ucar.edu/staff/rnair/remap.html
13. gx3v7_120309_pole.png
14. http://journals.ametsoc.org/doi/pdf/10.1175/JCLI3744.1
15. http://climate.lanl.gov/Models/POP/
16. ../modelnl/grid.html
17. ../modelnl/machines.html
18. http://subversion.tigris.org
19. http://svnbook.red-bean.com/en/1.5/svn.tour.cycle.html#svn.tour.cycle.resolve
20. https://svn-ccsm-release.cgd.ucar.edu
21. ../modelnl/compsets.html
22. ../modelnl/grid.html
23. ../modelnl/machines.html

Chapter 2. Creating and Setting Up A Case

The first step in creating a CESM experiment is to use **create_newcase**.

How to create a new case

CESM supports out of the box component sets[1], model grids[2] and hardware platforms[3]. Component sets (usually referred to as compsets) define both the specific model components that will be used in a given CESM configuration, *and* any component-specific namelist or configuration settings that are specific to this configuration. In the CESM1.2 release series (starting with CESM1.2.0) component sets and resolutions have been significantly changed to permit addressing the growing model complexity. Both compsets and models grids are now associated with three names: a new longname, a new short name (that is backwards compatible with the older CESM1.1 release series long name) and a new alias name (that is backwards compatible with the older CESM1.1 release series short name).

New Component Set Naming Convention

The new component set (compset) longname has the form

```
TIME_ATM[%phys]_LND[%phys]_ICE[%phys]_OCN[%phys]_ROF[%phys]_GLC[%phys]_WAV[%phys][_BGC%

TIME = model time period (e.g. 2000, 20TR, RCP8...)
ATM  = [CAM4, CAM5, DATM, SATM, XATM]
LND  = [CLM40, CLM45, DLND, SLND, XLND]
ICE  = [CICE, DICE, SICE, SICE]
OCN  = [POP2, DOCN, SOCN, XOCN,AQUAP]
ROF  = [RTM, DROF, SROF, XROF]
GLC  = [CISM1, SGLC, XGLC]
WAV  = [SWAV, XWAV]
BGC  = optional BGC scenario

The OPTIONAL %phys attributes specify sub-modes of the given system
For example DOCN%DOM is the  DOCN data ocean (rather than slab-ocean) mode.
ALL the possible %phys choices for each component are listed
by the calling create_newcase with the -list compsets argument.
ALL data models now have a %phys option that corresponds to the data
model mode
```

As an example, the new longname, 20TR_CAM4_CLM40%CN_CICE_POP2_RTM_SGLC_SWAV, refers to running the prognostic components CAM, CLM, RTM, CICE, POP2 with stubs SGLC and SWAV components. The particular configuration will be a transient 1850 to 2000 run using cam5 physics, clm4.0 physics with clm4.0 cn, prognostic cice (default) and pop2 (default). The shortname and alias for this compset are now B_1850-2000_CAM5_CN and B20TRC5CN, which correspond to the CESM1.1 series longname and shortname, respectively. *Any one of the three compset names* (longname, shortname or alias) can be used as input to **create_newcase**. It is now also much easier to create your own custom compset (see How do I create my own compset?). All the out-of-the-box CESM1.2 release series compsets are listed in component sets[4]. Upon clicking on any of the long names a pop up box will appear that provides more details of the component configuration.

New Overall Model Grid Naming Convention

The new model grid longname has the form

```
a%name_l%name_oi%name_r%name_m%mask_g%name_w%name
```

```
a%  = atmosphere grid
l%  = land grid
oi% = ocean/sea-ice grid (must be the same)
r%  = river grid
m%  = land mask grid
g%  = internal land-ice (CISM) grid
w%  = wave component grid (not relevant in CESM1.2 series)

From the point of view of model coupling - the glc (CISM) grid is assumed to
be identical to the land grid. However, the internal CISM grid can be different,
and is specified by the g% value.
```

As an example, the new longname, a%ne30np4_l%ne30np4_oi%gx1v6_r%r05_m%gx1v6_g%null_w%null, refers to running an ne30np4 spectral element 1-degree atmosphere and land grids a gx1v6 Greenland pole 1-degree ocean and sea-ice grids a 1/2 degree river routing grid, null wave and internal cism grids and an ocean/land mask that is determined by the gx1v6 ocean mask. The shortname and alias for this grid are now ne30np4_gx1v6 and ne30_g16, which correspond to the CESM1.1 series longname nad shortname respectively. *Any one of the three grid names* (longname, shortname or alias) can be used as input to **create_newcase**. It is now simpler for you to introduce new user defined grids (see Adding a new user-defined grid). All the out-of-the-box CESM1.2 release series model grids are listed in grids [5]. Upon clicking on any of the long names a pop up box will appear that provides more details of the model grid.

Component grids (such as the atmosphere grid or ocean grid above) are denoted by the following naming convention:

- "[dlat]x[dlon]" are regular lon/lat finite volume grids where dlat and dlon are the approximate grid spacing. The shorthand convention is "fnn" where nn is generally a pair of numbers indicating the resolution. An example is 1.9x2.5 or f19 for the approximately "2-degree" finite volume grid. Note that CAM uses an [nlat]x[nlon] naming convection internally for this grid.
- "Tnn" are spectral lon/lat grids where nn is the spectral truncation value for the resolution. The shorthand name is identical. An example is T85.
- "ne[X]np[Y]" are cubed sphere resolutions where X and Y are integers. The short name is generally ne[X]. An example is ne30np4 or ne30.
- "pt1" is a single grid point.
- "gx[D]v[n]" is a displaced pole grid where D is the approximate resolution in degrees and n is the grid version. The short name is generally g[D][n]. An example is gx1v6 or g16 for a grid of approximately 1-degree resolution.
- "tx[D]v[n]" is a tripole grid where D is the approximate resolution in degrees and n is the grid version.

Using create_newcase

You should first use the -h option in calling **create_newcase** to document its input options. In addition, the **create_newcase** -list [compsets,grids,machine] option can also be used to see which component sets, model grids, and machines are supported. The links above, however, provide a much more complete determination. **create_newcase** can be called with the following arguments:

```
create_newcase \
     -case case-name \
     -compset component-set \
```

```
-res resolution \
-mach machine-name \
[-compiler compiler-name> \
[-mpilib mpi-library-name] \
[-mach_dir alternative pathname for Machines directory] \
[-confopts [_AOA],[AOE],[_D],[_E],[_N],[_P],[_R]] \
[-pecount [S,M,L,X1,X2]] \
[-pes_file full-pathname] \
[-user_compset new user compset long name] \
[-user_grid_file full-pathname of user xml grid file] \
[-help [or -h]] |
[-list [compsets,grids,machines]  \
[-silent [or -s]] \
[-verbose [or -v]] \
[-xmlmode [normal, expert]] \
[-nowarning]
```

Required arguments to **create_newcase** are -case, -mach, -compset and -res. If you want to use your own pes setup file, specify the full pathname of that file for the optional -pes_file argument. The required pes_file format is provided at `$CCSMROOT/scripts/sample_pes_file.xml`.

Following is a simple example of using **create_newcase**: In what follows, $CCSMROOT is the full pathname of the root directory of the CESM distribution.

```
> cd $CCSMROOT/scripts
> create_newcase -case ~/cesm/example1 \
  -compset B_1850_CAM5_CN \
  -res ne30np4_gx1v6 \
  -mach yellowstone
```

This example creates a $CASEROOT directory `~/cesm1/example1` where $CASE is "example1" with a model resolution of 0.9x1.25_gx1v6 (a 1-degree atmosphere/land grid with a nominal 1-degree ocean/ice grid using the gx1v6 ocean mask). The component set B_1850_CN uses fully active components configured to produce a present-day simulation. The complete example appears in the basic example. $CASE can include letters, numbers, ".", and "_". Note that **create_newcase** creates the $CASEROOT directory. If the directory already exists, it prints a warning and aborts.

As a more general description, **create_newcase** creates the directory $CASEROOT, which is specified by the -case option. In $CASEROOT, **create_newcase** installs files to build and run the model and perform long term archiving of the case on the target platform. **create_newcase** also creates the directory `$CASEROOT/Buildconf/`, that in turn contains scripts to generate component namelist and build component libraries. The table below outlines the files and directories created by **create_newcase**

Table 2-1. Result of invoking create_newcase

Directory or Filename	Description
README.case	File detailing your **create_newcase** usage. This is a good place for you to keep track of runtime problems and changes.
CaseStatus	File containing a list of operations done in the current case.

Directory or Filename	Description
Buildconf/	Directory containing scripts to generate component namelists and component and utility libraries (e.g., PIO, MCT). You should never have to edit the contents of this directory (unlike in CESM1.0.5)
SourceMods/	Directory where you can place modified source code.
LockedFiles/	Directory that holds copies of files that should not be changed. The xml files are "locked" after its variables have been used by other parts of the system and cannot be changed. The scripts do this by "locking" a file and not permitting you to modify that file unless a 'clean' operation is performend. See the Section called *BASICS: Why is there file locking and how does it work?* in Chapter 6. You should never edit the contents of this directory.
Tools/	Directory containing support utility scripts. You should never need to edit the contents of this directory.
env_mach_specific	File used to set a number of machine-specific environment variables for building and/or running. Although you can edit this at any time, build environment variables should not be edited after a build is invoked.
env_case.xml	Sets case specific variables (e.g. model components, model and case root directories) and *cannot* be modified after a case has been created. To make changes, your should re-run **create_newcase** with different options.
env_build.xml	Sets model build settings, including component resolutions and component configuration options (e.g. CAM_CONFIG_OTPS) where applicable (see env_build.xml variables[6]).
env_mach_pes.xml	Sets component machine-specific processor layout (see the Section called *Changing the PE layout*). The settings in this are critical to a well-load-balanced simulation (see loadbalancing a run).
env_run.xml	Sets run-time settings such as length of run, frequency of restarts, output of coupler diagnostics, and short-term and long-term archiving. See run initialization variables[6], run stop variables[6], run restart control variables[6], for a more complete discussion of general run control settings.

Directory or Filename	Description
cesm_setup	Script used to set up the case (create the $CASE.run script and user_nl_xxx files)
$CASE.$MACH`.build`	Script to build component and utility libraries and model executable.
$CASE.$MACH`.clean_build`	Script to remove all object files and libraries and unlocks `Macros` and `env_build.xml`. This step is required before a clean build of the system.
$CASE.$MACH`.l_archive`	Script to performs long-term archiving of output data (see long-term archiving). This script will only be created if long-term archiving is available on the target machine.
xmlchange	Utility for modifying values in the xml files.
preview_namelists	Utility to enable users to see their component namelists in $CASEROOT`/CaseDocs` before running the model. NOTE: the namelists generated in $CASEROOT`/CaseDocs` should not be edited by the user - they are only their to document model behavior.
check_input_data	Utility that checks for various input datasets and moves them into place.
create_production_test	Utility used to create a test of your case.

For more complete information about the files in the case directory, see the Section called *BASICS: What are the directories and files in my case directory?* in Chapter 6

The xml variables in the `env_*.xml` files are translated into csh environment variables with the same name by the script **$CASEROOT/Tools/ccsm_getenv**. Conversion of xml variables to environment variables is used by numerous script utilities as part of building, and running a given case. It is important to note that you do not explicitly see this conversion.

Note: Users can only modify the xml variables. Users cannot modify the csh environment variables directly.

Complete lists of CESM environment variables in the xml files that appear in $CASEROOT are provided in case variables [6], pe layout variables [7], build-time variables [8], and run-time variables[9].

How to set up a case and customize the PE layout

Calling cesm_setup

The **cesm_setup** command does the following:

- Creates the Macros file if it does not exist. Calling **cesm_setup -clean** does not

remove this file.

- Creates the files, `user_nl_xxx`, (where xxx denotes the set of components targeted for the specific case). As an example, for a B_ compset, xxx would denote [cam,clm,rtm,cice,pop2,cpl]. *In CESM1.2, these files are where all user component namelist modifications are now made.* **cesm_setup -clean** does not remove these files
- Creates the file `$CASEROOT/$CASE.run` which runs the CESM model and performs short-term archiving of output data (see running CESM). **cesm_setup** also contains the necessary batch directives to run the model on the required machine for the requested PE layout. Any user modifications to `env_mach_pes.xml` must be done *before* **cesm_setup** is invoked. In the simplest case, **cesm_setup** can be run without modifying this file and default settings will be then be used. The **cesm_setup** command must be run in the $CASEROOT directory.

cesm_setup -clean moves `$CASEROOT/$CASE.run` and a copy of `env_mach_pes.xml` to a time-stamped directory in `MachinesHist`. The $CASEROOT directory will now appear as if **create_newcase** had just been run with the exception that already created Macros and user_nl_xxx files will not be touched and local modifications to the `env_*.xml` files will be preserved. After further modifications are made to `env_mach_pes.xml`, **cesm_setup** must be rerun before you can build and run the model.

If `env_mach_pes.xml` variables need to be changed after **cesm_setup** has been called, then **cesm_setup -clean** must be run first, followed by **cesm_setup**.

The following summarizes the new directories and files that are created by **cesm_setup**. For more information about the files in the case directory, see the Section called *BASICS: What are the directories and files in my case directory?* in Chapter 6.

Table 2-2. Result of calling cesm_setup

File or Directory	Description
Macros	File containing machine-specific makefile directives for your target platform/compiler. This is only created the *first* time that **cesm_setup** is called. Calling **cesm_setup -clean** will not remove the Macros file once it has been created.
user_nl_xxx[_NNNN] files	Files where all user modifications to component namelists are made. xxx denotes the set of components targeted for the specific case. NNNN goes from 0001 to the number of instances of that component (see the multiple instance discussion below). For example, for a B_ compset, xxx would denote [cam,clm,rtm,cice,pop2,cpl]. For a case where there is only 1 instance of each component (default) NNNN will not appear in the user_nl file names. A user_nl file of a given name will only be created once. Calling **cesm_setup -clean** will not remove any user_nl files. Changing the number of instances in the `env_mach_pes.xml` will only cause new user_nl files to be added to $CASEROOT.

File or Directory	Description
$CASE.run	File containing the necessary batch directives to run the model on the required machine for the requested PE layout. Runs the CESM model and performs short-term archiving of output data (see running CESM).
CaseDocs/	Directory that contains all the component namelists for the run. This is for reference only and files in this directory SHOULD NOT BE EDITED since they will be overwritten at build time and run time.
env_derived	File containing environmental variables derived from other settings. Should *not* be modified by the user.

Changing the PE layout

`env_mach_pes.xml`[10] variables determine the number of processors for each component, the number of instances of each component and the layout of the components across the hardware processors. Optimizing the throughput and efficiency of a CESM experiment often involves customizing the processor (PE) layout for load balancing. CESM has significant flexibility with respect to the layout of components across different hardware processors. In general, the CESM components -- atm, lnd, ocn, ice, glc, rof, and cpl -- can run on overlapping or mutually unique processors. Whereas Each component is associated with a unique MPI communicator, the driver runs on the union of all processors and controls the sequencing and hardware partitioning. The component processor layout is via three settings: the number of MPI tasks, the number of OpenMP threads per task, and the root MPI processor number from the global set.

For example, the following `env_mach_pes.xml` settings

```
<entry id="NTASKS_OCN" value="128" />
<entry id="NTHRDS_OCN" value="1" />
<entry id="ROOTPE_OCN" value="0" />
```

would cause the ocean component to run on 128 hardware processors with 128 MPI tasks using one thread per task starting from global MPI task 0 (zero).

In this next example:

```
<entry id="NTASKS_ATM" value="16" />
<entry id="NTHRDS_ATM" value="4"  />
<entry id="ROOTPE_ATM" value="32" />
```

the atmosphere component will run on 64 hardware processors using 16 MPI tasks and 4 threads per task starting at global MPI task 32. There are NTASKS, NTHRDS, and ROOTPE input variables for every component in `env_mach_pes.xml`. There are some important things to note.

- NTASKS must be greater or equal to 1 (one) even for inactive (stub) components.
- NTHRDS must be greater or equal to 1 (one). If NTHRDS is set to 1, this generally means threading parallelization will be off for that component. NTHRDS should never be set to zero.

- The total number of hardware processors allocated to a component is NTASKS * NTHRDS.
- The coupler processor inputs specify the pes used by coupler computation such as mapping, merging, diagnostics, and flux calculation. This is distinct from the driver which always automatically runs on the union of all processors to manage model concurrency and sequencing.
- The root processor is set relative to the MPI global communicator, not the hardware processors counts. An example of this is below.
- The layout of components on processors has no impact on the science. The scientific sequencing is hardwired into the driver. Changing processor layouts does not change intrinsic coupling lags or coupling sequencing. ONE IMPORTANT POINT is that for a fully active configuration, the atmosphere component is hardwired in the driver to never run concurrently with the land or ice component. Performance improvements associated with processor layout concurrency is therefore constrained in this case such that there is never a performance reason not to overlap the atmosphere component with the land and ice components. Beyond that constraint, the land, ice, coupler and ocean models can run concurrently, and the ocean model can also run concurrently with the atmosphere model.
- If all components have identical NTASKS, NTHRDS, and ROOTPE set, all components will run sequentially on the same hardware processors.

The root processor is set relative to the MPI global communicator, not the hardware processor counts. For instance, in the following example:

```
<entry id="NTASKS_ATM" value="16" />
<entry id="NTHRDS_ATM" value="4"  />
<entry id="ROOTPE_ATM" value="0"  />
<entry id="NTASKS_OCN" value="64" />
<entry id="NTHRDS_OCN" value="1"  />
<entry id="ROOTPE_OCN" value="16" />
```

the atmosphere and ocean are running concurrently, each on 64 processors with the atmosphere running on MPI tasks 0-15 and the ocean running on MPI tasks 16-79. The first 16 tasks are each threaded 4 ways for the atmosphere. The batch submission script ($CASE.run) should automatically request 128 hardware processors, and the first 16 MPI tasks will be laid out on the first 64 hardware processors with a stride of 4. The next 64 MPI tasks will be laid out on the second set of 64 hardware processors.

If you set ROOTPE_OCN=64 in the preceding example, then a total of 176 processors would have been requested and the atmosphere would have been laid out on the first 64 hardware processors in 16x4 fashion, and the ocean model would have been laid out on hardware processors 113-176. Hardware processors 65-112 would have been allocated but completely idle.

> **Note:** `env_mach_pes.xml` *cannot* be modified after "./cesm_setup" has been invoked without first invoking "cesm_setup -clean". For an example of changing pes, see the Section called *BASICS: How do I change processor counts and component layouts on processors?* in Chapter 6

Multi-instance component functionality

Like the CESM1.1 series, the CESM1.2 series also has the new capability to run multiple component instances under one model executable. The only caveat to this usage is that if N multiple instances of any one active component is used, then N multiple instances of ALL active components are required. More details are discussed below. The primary motivation for this development was to be able to run an ensemble

Kalman-Filter for data assimilation and parameter estimation (e.g. UQ). However, it also provides you with the ability to run a set of experiments within a single CESM executable where each instance can have a different namelist, and have all the output go to one directory.

In the following an F compset will be used as an illustration. Utilizing the multiple instance code involves the following steps:

1. create the case

```
> create_newcase -case Fmulti -compset F  -res ne30_g16 -mach hopper
> cd Fmulti
```

2. Lets assume the following out of the box pe-layout for hopper:

```
<entry id="NTASKS_ATM"   value="128"  />
<entry id="NTHRDS_ATM"   value="1"  />
<entry id="ROOTPE_ATM"   value="0"  />
<entry id="NINST_ATM"   value="1"  />
<entry id="NINST_ATM_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_LND"   value="128"  />
<entry id="NTHRDS_LND"   value="1"  />
<entry id="ROOTPE_LND"   value="0"  />
<entry id="NINST_LND"   value="1"  />
<entry id="NINST_LND_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_ICE"   value="128"  />
<entry id="NTHRDS_ICE"   value="1"  />
<entry id="ROOTPE_ICE"   value="0"  />
<entry id="NINST_ICE"   value="1"  />
<entry id="NINST_ICE_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_OCN"   value="128"  />
<entry id="NTHRDS_OCN"   value="1"  />
<entry id="ROOTPE_OCN"   value="0"  />
<entry id="NINST_OCN"   value="1"  />
<entry id="NINST_OCN_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_GLC"   value="128"  />
<entry id="NTHRDS_GLC"   value="1"  />
<entry id="ROOTPE_GLC"   value="0"  />
<entry id="NINST_GLC"   value="1"  />
<entry id="NINST_GLC_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_WAV"   value="128"  />
<entry id="NTHRDS_WAV"   value="1"  />
<entry id="ROOTPE_WAV"   value="0"  />
<entry id="NINST_WAV"   value="1"  />
<entry id="NINST_WAV_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_CPL"   value="128"  />
<entry id="NTHRDS_CPL"   value="1"  />
<entry id="ROOTPE_CPL"   value="0"  />
```

For an F compset, only atm, lnd, rof are full prognostic components. The ocn is a prescribed data component, cice is a mixed prescribed/prognostic component (ice-coverage is prescribed) and glc and wav are stub components. Lets say we want to run 2 instances of CAM in this experiment. The current implementation of multi-instances will also require you to run 2 instances of CLM, CICE and RTM. However, you have the flexibility to run either 1 or 2 instances of DOCN (we can ignore glc and wav since they do not do anything in this compset). To run 2 instances of CAM, CLM, CICE, RTM and DOCN, all you need to do is to change NINST_ATM, NINST_LND, NINST_ICE, NINST_ROF and NINST_DOCN above from 1 to 2. This will result in the following `env_mach_pes.xml` file:

```
<entry id="NTASKS_ATM"   value="128"  />
<entry id="NTHRDS_ATM"   value="1"  />
<entry id="ROOTPE_ATM"   value="0"  />
<entry id="NINST_ATM"   value="2"  />
<entry id="NINST_ATM_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_LND"   value="128"  />
<entry id="NTHRDS_LND"   value="1"  />
<entry id="ROOTPE_LND"   value="0"  />
<entry id="NINST_LND"   value="2"  />
<entry id="NINST_LND_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_ICE"   value="128"  />
<entry id="NTHRDS_ICE"   value="1"  />
<entry id="ROOTPE_ICE"   value="0"  />
<entry id="NINST_ICE"   value="2"  />
<entry id="NINST_ICE_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_ROF"   value="128"  />
<entry id="NTHRDS_ROF"   value="1"  />
<entry id="ROOTPE_ROF"   value="0"  />
<entry id="NINST_ROF"   value="2"  />
<entry id="NINST_ROF_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_OCN"   value="128"  />
<entry id="NTHRDS_OCN"   value="1"  />
<entry id="ROOTPE_OCN"   value="0"  />
<entry id="NINST_OCN"   value="2"  />
<entry id="NINST_OCN_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_GLC"   value="128"  />
<entry id="NTHRDS_GLC"   value="1"  />
<entry id="ROOTPE_GLC"   value="0"  />
<entry id="NINST_GLC"   value="1"  />
<entry id="NINST_GLC_LAYOUT"   value="concurrent"  />

<entry id="NTASKS_CPL"   value="128"  />
<entry id="NTHRDS_CPL"   value="1"  />
<entry id="ROOTPE_CPL"   value="0"  />
```

As a result of this, you will have 2 instances of CAM, CLM and CICE (prescribed) and RTM, each running concurrently on 64 MPI tasks - and only 1 instance of DOCN.

3. A separate user_nl_xxx_NNNN file (where NNNN is the number of the component instances) will be generated when **cesm_setup** is called. In particular, calling **cesm_setup** with the above `env_mach_pes.xml` file will result in the following `user_nl_*` files in $CASEROOT

```
user_nl_cam_0001
user_nl_cam_0002
user_nl_cice_0001
user_nl_cice_0002
user_nl_clm_0001
user_nl_clm_0002
user_nl_cpl
user_nl_docn_0001
user_nl_docn_0002
user_nl_rtm_0001
user_nl_rtm_0002
```

and the following `*_in_*` files and `*txt*` files in $CASEROOT/CaseDocs:

```
atm_in_0001
atm_in_0002
docn.streams.txt.prescribed_0001
docn.streams.txt.prescribed_0002
```

```
docn_in_0001
docn_in_0002
docn_ocn_in_0001
docn_ocn_in_0002
drv_flds_in
drv_in
ice_in_0001
ice_in_0002
lnd_in_0001
lnd_in_0002
rof_in_0001
rof_in_0002
```

The namelist for each component instance can be modified by changing the corresponding user_nl_xxx_NNNN file for that component instance. Modifying the user_nl_cam_0002 will result in the namelist changes you put in to be active ONLY for instance 2 of CAM. To change the DOCN stream txt file instance 0002, you should place a copy of `docn.streams.txt.prescribed_0002` in $CASEROOT with the name `user_docn.streams.txt.prescribed_0002` and modify it accordlingly.

It is also important to stress the following points:

1. **Different component instances can ONLY differ by differences in namelist settings - they are ALL using the same model executable.**
2. Only 1 coupler component is supported in the CESM1.2 series multiple instance implementation.
3. `user_nl_*` files once they are created by **cesm_setup** *ARE NOT* removed by calling **cesm_setup -clean**. The same is true for `Macros` files.
4. In general, you should run multiple instances concurrently (the default setting in `env_mach_pes.xml`). The serial setting is only for EXPERT USERS in upcoming development code implementations.

Modifying an xml file

You can edit the xml files directly to change the variable values. However, modification of the xml variables is best done using **xmlchange** in the $CASEROOT directory since it performs variable error checking as part of changing values in the xml files. To invoke **xmlchange**:

```
xmlchange <entry id>=<value>
-- OR --
xmlchange -id <entry id> -val <name> -file <filename>
          [-help] [-silent] [-verbose] [-warn] [-append] [-file]
```

`-id`

The xml variable name to be changed. (required)

`-val`

The intended value of the variable associated with the -id argument. (required)

Note: If you want a single quotation mark ("'", also called an apostrophe) to appear in the string provided by the -val option, you must specify it as "'".

`-file`

The xml file to be edited. (optional)

`-silent`

Turns on silent mode. Only fatal messages will be issued. (optional)

`-verbose`

Echoes all settings made by **create_newcase** and **cesm_setup**. (optional)

`-help`

Print usage info to STDOUT. (optional)

Cloning a case (Experts only)

This is an advanced feature provided for expert users. If you are a new user, skip this section.

If you have access to the run you want to clone, the **create_clone** command will create a new case while also preserving local modifications to the case that you want to clone. You can run the utility **create_clone** either from $CCSMROOT or from the directory where you want the new case to be created. It has the following arguments:

`-case`

The name or path of the new case.

`-clone`

The full pathname of the case to be cloned.

`-silent`

Enables silent mode. Only fatal messages will be issued.

`-verbose`

Echoes all settings.

`-help`

Prints usage instructions.

Here is the simplest example of using **create_clone**:

```
> cd $CCSMROOT/scripts
> create_clone -case $CASEROOT -clone $CLONEROOT
```

create_clone will preserve any local namelist modifications made in the user_nl_xxxx files as well as any source code modifications in the SourceMods tree. Note that the new case directory will be identical to the cloned case directory *except* for the original cloned scripts $CASEROOT.$MACH.build, $CASEROOT.$MACH.clean_build, $CASEROOT.$MACH.run, and $CASEROOT.$MACH.l_archive, which will have new names in the new case.

Important:: Do not change anything in the `env_case.xml` file. The $CASEROOT/ directory will now appear as if **create_newcase** had just been run -- with the exception that local modifications to the env_* files are preserved.

Another approach to duplicating a case is to use the information in that case's README.case file to create a new case. Note that this approach will *not* preserve

any local modifications that were made to the original case, such as source-code or build-script modifications; you will need to import those changes manually.

Notes

1. ../modelnl/compsets.html
2. ../modelnl/grid.html
3. ../modelnl/machines.html
4. ../modelnl/compsets.html
5. ../modelnl/grid.html
6. ../modelnl/env_case.html
7. ../modelnl/env_mach_pes.html
8. ../modelnl/env_build.html
9. ../modelnl/env_run.html
10. ../modelnl/env_mach_pes.html

Chapter 3. Building CESM

The following summarizes details of building the model exectuable.

How do I build my model?

After calling **cesm_setup**, you can build the model executable by running **./$CASE.build**. Running this will:

1. create the component namelists in $RUNDIR (by calling the `Buildconf/$component.buildnml.csh` scripts).
2. check for the required input data sets and download missing data automatically on local disk, and if successful proceed to the following steps.
3. create the necessary utility libraries by calling `Buildconf/mct.buildlib`, `Buildconf/pio.buildlib` and `Buildconf/gptl.buildlib` and `Buildconf/csm_share.buildlib`.
4. create the necessary component libraries by calling `Buildconf/$component.buildexe.csh`, where $component is the name of atm, lnd, rof, ocn, cice, glc and cpl components (which depends on the compset being used).
5. create the model executable by calling `Buildconf/cesm.buildexe.csh`.

`$CASEROOT/Tools/Makefile` and `$CASEROOT/Macros` (generated by calling **cesm_setup**) are used to generate the utility and component libraries and the model executable. You do not need to change the default build settings to create the executable. However, since the CESM scripts provide you with a great deal of flexibility in customizing various aspects of the build process, it is useful to become familiar with these in order to make optimal use of the system.

The `env_build.xml` variables[1], control various aspects of building the model executable. Most of the variables should not be modified by users. Among the variables that you can modify are EXEROOT, RUNDIR, BUILD_THREADED, DEBUG and GMAKE_J. Full documentation for each variable is provided in The `env_build.xml` variables[2].

```
> cd $CASEROOT
> ./$CASE.build
```

Diagnostic comments will appear as the build proceeds. The following line indicates that the component namelists have been generated successfully:

```
....
CCSM BUILDNML SCRIPT HAS FINISHED SUCCESSFULLY
....
```

When the required case input data in $DIN_LOC_ROOT has been successfully checked, you will see:

```
CCSM PRESTAGE SCRIPT STARTING
...
CCSM PRESTAGE SCRIPT HAS FINISHED SUCCESSFULLY
```

Finally, the build script generates the utility and component libraries and the model executable. There should be a line for the mct, pio, and gptl libraries, as well as each of the components. Each is date stamped, and a pointer to the build log file for that library or component is shown. Successful completion is indicated by:

```
CCSM BUILDEXE SCRIPT HAS FINISHED SUCCESSFULLY
```

The build log files have names of the form $model.bldlog.$datestamp and are located in $RUNDIR. If they are compressed (indicated by a .gz file extension), then the build ran successfully.

Invoking **$CASE.build** creates the following directory structure in $EXEROOT:

```
$EXEROOT/atm
$EXEROOT/cesm
$EXEROOT/cpl
$EXEROOT/csm_share
$EXEROOT/glc
$EXEROOT/ice
$EXEROOT/lib
$EXEROOT/lnd
$EXEROOT/mct
$EXEROOT/ocn
$EXEROOT/pio
$EXEROOT/rof
```

The atm/, cesm/, cpl/, glc/, ice/, lnd/, ocn/ and rof/ subdirectories in $EXEROOT each contain an 'obj/' directory where the compiled object files for the target model component is placed. These object files are collected into libraries that are placed in 'lib/' along with the mct/mpeu, pio, gptl, and csm_share libraries. Special include modules are also placed in lib/include. The model executable 'cesm.exe' is placed directly in $EXEROOT. On the other hand, component namelists, component logs, output datasets, and restart files are placed in $RUNDIR. It is important to note that in CESM $RUNDIR and $EXEROOT are independent variables which are set in the file `config_machines.xml` in the directory `$CCSMROOT/scripts/ccsm_utils/Machines/`.

Input data

All active and data components use input datasets. A local disk needs $DIN_LOC_ROOT to be populated with input data in order to run CESMwith these components. For all machines, input data is provided as part of the release via data from the CESM subversion input data server. However, on supported machines (and some non-supported machines), data already exists in the default local filesystem input data area as specified by $DIN_LOC_ROOT (see below).

Input data is handled by the build process as follows:

- The buildnml scripts in `Buildconf/i` create listings of required component input datasets in the `Buildconf/$component.input_data_list` files.
- `$CASE.build` checks for the presence of the required input data files in the root directory $DIN_LOC_ROOT. If all required data sets are found on local disk, then the build can proceed.
- If any of the required input data sets are not found, the build script will abort and the files that are missing will be listed. At this point, you must obtain the required data from the input data server using **check_input_data** with the -export option.

The `env_run.xml` variables DIN_LOC_ROOT and DIN_LOC_ROOT_CLMFORC determine where you should expect input data to reside on local disk. See the input data variables[3].

User-created input data

If you want to use new user-created dataset(s) and give these dataset(s) names that are different than the names in $DIN_LOC_ROOT, we recommend using the script **link_dirtree** in the directory `$CCSMROOT/scripts`. **link_dirtree** creates a virtual copy of the input data directory by linking one directory tree to another. The full directory

structure of the original directory is duplicated and the files are linked. To use this script, use the -h optiion for usage.

```
> cd $CCSMROOT/scripts
> ./link_dirtree -h
```

link_dirtree can be conveniently used to generate the equivalent of a local copy of $DIN_LOC_ROOT which can then be populated with user-specified input datasets. For example, you can first generate a virtual copy of $DIN_LOC_ROOT in /user/home/newdata with the following command:

```
> link_dirtree $DIN_LOC_ROOT /user/home/newdata
```

then incorporate the new dataset(s) directly into the appropriate directory in /user/home/newdata.

Using the input data server

The script `$CASEROOT/check_input_data` determines if the required data files for the case exist on local disk in the appropriate subdirectory of $DIN_LOC_ROOT. If any of the required datasets do not exist locally, **check_input_data** provides the capability for downloading them to the $DIN_LOC_ROOT directory hierarchy via interaction with the input data server. You can independently verify that the required data is present locally by using the following commands:

```
> cd $CASEROOT
> check_input_data -help
> check_input_data -inputdata $DIN_LOC_ROOT -check
```

If input data sets are missing, you must obtain the datasets from the input data server:

```
> cd $CASEROOT
> check_input_data -inputdata $DIN_LOC_ROOT -export
```

Required data files not on local disk will be downloaded through interaction with the Subversion input data server. These will be placed in the appropriate subdirectory of $DIN_LOC_ROOT. For what to expect when interacting with a Subversion repository, see downloading input data.

Rebuilding the model

You should rebuild the model under the following circumstances:

If either `env_build.xml` or `Macros` has been modified, and/or if code is added to `SourceMods/src.*`, then it's safest to clean the build and rebuild from scratch as follows,

```
> cd $CASEROOT
> ./$CASE.clean_build
> ./$CASE.build
```

If you have ONLY modified the PE layout in `env_mach_pes.xml` (see setting the PE layout) then it's possible that a clean is not required.

```
> cd $CASEROOT
> $CASE.build
```

But if the threading has been turned on or off in any component relative to the previous build, then the build script should fail with the following error

```
ERROR SMP STATUS HAS CHANGED
  SMP_BUILD = a0l0i0o0g0c0
  SMP_VALUE = a1l0i0o0g0c0
  A manual clean of your obj directories is strongly recommendend
  You should execute the following:
    ./b39pA1.yellowstone.clean_build
  Then rerun the build script interactively
  ---- OR ----
  You can override this error message at your own risk by executing
    ./xmlchange SMP_BUILD=0
  Then rerun the build script interactively
```

and suggest that the model be rebuilt from scratch.

You are responsible for manually rebuilding the model when needed. If there is any doubt, you should rebuild.

Notes

1. ../modelnl/env_build.html
2. ../modelnl/env_build.html
3. ../modelnl/env_run.html#run_din

Chapter 4. Running CESM

To run a CESM case, you must submit the batch script `$CASE.run`. In addition, you also need to modify `env_run.xml` for your particular needs.

The `env_run.xml` file[1] contains variables which may be modified at the initialization of a model run and during the course of that model run. These variables comprise coupler namelist settings for the model stop time, model restart frequency, coupler history frequency and a flag to determine if the run should be flagged as a continuation run. In general, you only need to set the variables $STOP_OPTION and $STOP_N. The other coupler settings will then be given consistent and reasonable default values. These default settings guarantee that restart files are produced at the end of the model run.

Customizing runtime settings

In the following, we focus on the handling of run control (e.g. length of run, continuing a run) and output data. We also give a more detailed description of CESM restarts.

Controlling starting, stopping and restarting a run

The case initialization type is set in `env_run.xml`. A CESM run can be initialized in one of three ways; startup, branch, or hybrid.

startup

In a startup run (the default), all components are initialized using baseline states. These baseline states are set independently by each component and can include the use of restart files, initial files, external observed data files, or internal initialization (i.e., a "cold start"). In a startup run, the coupler sends the start date to the components at initialization. In addition, the coupler does not need an input data file. In a startup initialization, the ocean model does not start until the second ocean coupling (normally the second day).

branch

In a branch run, all components are initialized using a consistent set of restart files from a previous run (determined by the $RUN_REFCASE and $RUN_REFDATE variables in `env_run.xml`). The case name is generally changed for a branch run, although it does not have to be. In a branch run, setting $RUN_STARTDATE is ignored because the model components obtain the start date from their restart datasets. Therefore, the start date cannot be changed for a branch run. This is the same mechanism that is used for performing a restart run (where $CONTINUE_RUN is set to TRUE in the `env_run.xml` file).

Branch runs are typically used when sensitivity or parameter studies are required, or when settings for history file output streams need to be modified while still maintaining bit-for-bit reproducibility. Under this scenario, the new case is able to produce an exact bit-for-bit restart in the same manner as a continuation run *if* no source code or component namelist inputs are modified. All models use restart files to perform this type of run. $RUN_REFCASE and $RUN_REFDATE are required for branch runs.

To set up a branch run, locate the restart tar file or restart directory for $RUN_REFCASE and $RUN_REFDATE from a previous run, then place those files in the $RUNDIR directory. See setting up a branch run for an example.

hybrid

A hybrid run indicates that CESM is initialized more like a startup, but uses initialization datasets *from a previous case*. This is somewhat analogous to a branch run with relaxed restart constraints. A hybrid run allows users to bring together combinations of initial/restart files from a previous case (specified by $RUN_REFCASE) at a given model output date (specified by $RUN_REFDATE). Unlike a branch run, the starting date of a hybrid run (specified by $RUN_STARTDATE) can be modified relative to the reference case. In a hybrid run, the model does not continue in a bit-for-bit fashion with respect to the reference case. The resulting climate, however, should be continuous provided that no model source code or namelists are changed in the hybrid run. In a hybrid initialization, the ocean model does not start until the second ocean coupling (normally the second day), and the coupler does a "cold start" without a restart file.

The variable $RUN_TYPE determines the initialization type. This setting is only important for the initial run of a production run when the $CONTINUE_RUN variable is set to FALSE. After the initial run, the $CONTINUE_RUN variable is set to TRUE, and the model restarts exactly using input files in a case, date, and bit-for-bit continuous fashion. The variable $RUN_TYPE is the start date (in yyyy-mm-dd format) either a startup or hybrid run. If the run is targeted to be a hybrid or branch run, you must also specify values for $RUN_REFCASE and $RUN_REFDATE. All run startup variables are discussed in run start control variables[2].

Before a job is submitted to the batch system, you need to first check that the batch submission lines in `$CASE.run` are appropriate. These lines should be checked and modified accordingly for appropriate account numbers, time limits, and stdout/stderr file names. You should then modify `env_run.xml` to determine the key run-time settings. See controlling run termination[3], controlling run restarts[4], and performing model restarts for more details. A brief note on restarting runs. When you first begin a branch, hybrid or startup run, CONTINUE_RUN must be set to FALSE. When you successfully run and get a restart file, you will need to change CONTINUE_RUN to TRUE for the remainder of your run. See performing model restarts for more details.

By default,

```
STOP_OPTION = ndays
STOP_N = 5
STOP_DATE = -999
```

The default setting is only appropriate for initial testing. Before a longer run is started, update the stop times based on the case throughput and batch queue limits. For example, if the model runs 5 model years/day, set RESUBMIT=30, STOP_OPTION= nyears, and STOP_N= 5. The model will then run in five year increments, and stop after 30 submissions.

Customizing component-specific namelist settings

In your $CASEROOT directory, the subdirectory `$CASEROOT/Buildconf` contains files to create the component namelists, build the component libraries and create the model executable. `Buildconf/$component.buildexe.csh` creates the component libraries and `Buildconf/$component.buildnml.csh` creates the component namelists. A new feature in the CESM1.1 and CESM1.2 release series is that *ALL* CESM components now use a component-specific **build-namelist** utility (similar to that of CAM, CLM and CICE in the CESM1.0 series) to generate their respective model namelists. In addition, CAM, CLM and CICE have an associated **configure** utility that sets up compile time configuration options and is also called from the corresponding Buildconf/*.buildnml.csh (e.g. Buildconf/cam.buildnml.csh).

In the CESM1.2 series, user specific component namelist changes should only be made only by editing the `$CASEROOT/user_nl_xxx` files OR by changing xml variables in env_run.xml[5] or env_build.xml[6]. A full discussion of how to change the namelist variables for each component is provided below. You can preview the case component namelists by running `preview_namelists` in your $CASEROOT. Calling `preview_namelists` results in the creation of component namelists (e.g. atm_in, lnd_in, .etc) in `$CASEROOT/CaseDocs/`. A complete documentation of all model component namelists [7] for CESM1.2 releases is now available. The namelist files created in the `CaseDocs/` are there only for user reference and *SHOULD NOT BE EDITED* since they are overwritten every time `preview_namelists`, `$CASE.run` and `$CASE.build` are called. In CESM1.2, (like CESM1.1 but unlike CESM1.0) the only files that you should modify are in $CASEROOT. No files in Buildconf/ should be changed. The following represents a summary of controlling and modifying component-specific run-time settings:

DRV

In CESM1.2, driver namelist[8] are in two groups - those that are set directly from xml variables in env_case.xml[9], env_mach_pes.xml[10] and env_run.xml[11], and those that are set by the driver build-namelist utility (`$CCSMROOT/models/drv/bld/build-namelist`) for the target compset and resolution. Except for the following driver namelist variables (see below), driver namelist variables that are in `env_run.xml` can be changed either by changing the xml variable OR by adding the correct key-word value pair at the end of `user_nl_cpl`, where any changes in `user_nl_cpl` *will take precedence* over values set in the xml file. For example, to change eps_frac to 1.0e-15, add the following line to the end of the `user_nl_cpl`, "eps_frac = 1.0e-15". To see the result of this modification to `user_nl_cpl` call `preview_namelists` and verify that this new value appears in `CaseDocs/drv_in`.

```
The following namelist variables MAY NOT be changed in user_nl_cpl -
but must be changed in the appropriate $CASEROOT xml file.
XXX  refers to ATM,LND,ICE,OCN,ROF,GLC,WAV
=====================================
drv namelist   => xml variable
variable
=====================================
case_name      => CASE
username       => CCSMUSER
hostname       => MACH
model_version  => CCSM_REPOTAG
start_type     => RUN_TYPE
start_ymd      => RUN_STARTDATE
start_tod      => START_TOD
XXX_cpl_dt     => XXX_NCPL
XXX_ntasks     => NTASKS_XXX
XXX_nthreads   => NTHRDS_XXX
XXX_rootpe     => ROOTPE_XXX
XXX_pestride   => PSTRID_XXX
XXX_layout     => NINST_XXX_LAYOUT
```

CAM

CAM's configure [12] and build-namelist [13] utilities are called by `Buildconf/cam.buildnml.csh`. CAM_CONFIG_OPTS[14], CAM_NAMELIST_OPTS[15] and CAM_NML_USECASE[16] are used to set compset variables (e.g., "-phys cam5" for CAM_CONFIG_OPTS) and in general should not be modified for supported compsets. For a complete documentation of namelist settings, see CAM namelist variables[17]. To modify CAM namelist settings, you should add the appropriate keyword/value pair at the end of the `$CASEROOT/user_nl_cam` file (see the documentation for each file at the top of that file). For example, to change the solar constant to 1363.27, modify the user_nl_cam file to contain the following line at the end "solar_const=1363.27".

To see the result of adding this, call **preview_namelists** and verify that this new value appears in `CaseDocs/atm_in`.

CLM

CLM's configure[18] and build-namelist[19] utilities are called by `Buildconf/clm.buildnml.csh`. CLM_CONFIG_OPTS[20] and CLM_NML_USE_CASE[21] are used to set compset specific variables and in general should not be modified for supported compsets. For a complete documentation of namelist settings, see CLM namelist variables[22]. To modify CLM namelist settings, you should add the appropriate keyword/value pair at the end of the `$CASEROOT/user_nl_clm` file (see the documentation for each file at the top of that file). To see the result of your change, call **preview_namelists** and verify that the changes appear correctly in `CaseDocs/lnd_in`.

RTM

RTM's **build-namelist** utility is called by `Buildconf/rtm.buildnml.csh`. For a complete documentation of namelist settings, see RTM namelist variables[23]. To modify RTM namelist settings, you should add the appropriate keyword/value pair at the end of the `$CASEROOT/user_nl_rtm` file (see the documentation for each file at the top of that file). To see the result of your change, call **preview_namelists** and verify that the changes appear correctly in `CaseDocs/rof_in`.

CICE

CICE's configure[24] and build-namelist[25] utilities are now called by `Buildconf/cice.buildnml.csh`. Note that CICE_CONFIG_OPTS[26], and CICE_NAMELIST_OPTS[27] are used to set compset specific variables and in general should not be modified for supported compsets. For a complete documentation of namelist settings, see CICE namelist variables[28]. To modify CICE namelist settings, you should add the appropriate keyword/value pair at the end of the `$CASEROOT/user_nl_cice` file (see the documentation for each file at the top of that file). To see the result of your change, call **preview_namelists** and verify that the changes appear correctly in `CaseDocs/ice_in`.

In addition, **cesm_setup** creates CICE's compile time block decomposition variables[29] in `env_build.xml` as follows:

```
./cesm_setup
   ⇓
Buildconf/cice.buildnml.csh and $NTASKS_ICE and $NTHRDS_ICE
   ⇓
env_build.xml variables CICE_BLCKX, CICE_BLCKY, CICE_MXBLCKS, CICE_DECOMPTYPE
CPP variables in cice.buildexe.csh
```

POP2

See POP2 namelist variables[30] for a complete description of the POP2 run-time namelist variables. Note that OCN_COUPLING, OCN_ICE_FORCING, OCN_TRANSIENT[31] are normally utilized ONLY to set compset specific variables and should not be edited. For a complete documentation of namelist settings, see CICE namelist variables[32]. To modify POP2 namelist settings, you should add the appropriate keyword/value pair at the end of the `$CASEROOT/user_nl_pop2` file (see the documentation for each file at the top of that file). To see the result of your change, call **preview_namelists** and verify that the changes appear correctly in `CaseDocs/ocn_in`.

In addition, **cesm_setup** also generates POP2's compile time compile time block decomposition variables[33] in `env_build.xml` as follows:

```
./cesm_setup
   ⇓
```

```
Buildconf/pop2.buildnml.csh and $NTASKS_OCN and $NTHRDS_OCN
   ⇓
env_build.xml variables POP2_BLCKX, POP2_BLCKY, POP2_MXBLCKS, POP2_DECOMPTYPE
CPP variables in pop2.buildexe.csh
```

CISM

See CISM namelist variables[34] for a complete description of the CISM run-time namelist variables. This includes variables that appear both in `cism_in` and in `cism.config`. To modify any of these settings, you should add the appropriate keyword/value pair at the end of the `user_nl_cism` file (see the documentation for each file at the top of that file). To see the result of your change, call **preview_namelists** and verify that the changes appear correctly in `CaseDocs/cism_in` and `CaseDocs/cism.config`.

There are also some run-time settings set via `env_run.xml`, as documented in CISM run time variables[35] - in particular, the model resolution, set via CISM_GRID. The value of CISM_GRID determines the default value of a number of other namelist parameters.

DATM

DATM is discussed in detail in Data Model's User's Guide[36]. DATM is normally used to provide observational forcing data (or forcing data produced by a previous run using active components) to drive CLM (I compset), POP2 (C compset), and POP2/CICE (G compset). As a result, DATM variable settings are specific to the compset that will be targeted.

DATM can be user configured in three different ways.

You can set DATM run-time variables[37] my modifying control settings for CLM and CPLHIST forcing.

You can edit `user_nl_datm` to change namelist settings namelists settings by adding all user specific namelist changes in the form of "namelist_var = new_namelist_value". Note that any namelist variable from shr_strdata_nml and datm_nml can be modified below using the this syntax. Use **preview_namelists** to view (not modify) the output namelist in `CaseDocs`.

You can modify the contents of a DATM stream txt file. To do this:

- use **preview_namelists** to obtain the contents of the stream txt files in `CaseDocs`
- place a *copy* of this file in $CASEROOT with the string *"user_"* prepended
- **Make sure you change the permissions of the file to be writeable** (chmod 644)
- Modify the `user_datm.streams.txt.*` file.

As an example, if the stream txt file in `CaseDocs/` is `datm.streams.txt.CORE2_NYF.GISS`, the modified copy in $CASEROOT should be `user_datm.streams.txt.CORE2_NYF.GISS`. After calling **preview_namelists** again, you should see your new modifications appear in `CaseDocs/datm.streams.txt.CORE2_NYF.GISS`.

DOCN

DOCN is discussed in detail in Data Model's User's Guide[38].

DOCN running in prescribed mode assumes that the only field in the input stream is SST and also that SST is in Celsius and must be converted to Kelvin. All other fields are set to zero except for ocean salinity, which is set to a constant reference salinity value. Normally the ice fraction data (used for prescribed CICE) is found in the same data files that provide SST data to the data ocean model since SST and ice fraction data are derived from the same observational data sets and are consistent with each other. For DOCN prescribed mode, default yearly

climatological datasets are provided for various model resolutions. For multi-year runs requiring AMIP datasets of sst/ice_cov fields, you need to set the variables for DOCN_SSTDATA_FILENAME, DOCN_SSTDATA_YEAR_START, and DOCN_SSTDATA_YEAR_END[39]. CICE in prescribed mode also uses these values.

DOCN running as a slab ocean model is used (in conjunction with CICE running in prognostic mode) in all E compsets. SOM ("slab ocean model") mode is a prognostic mode. This mode computes a prognostic sea surface temperature and a freeze/melt potential (surface Q-flux) used by the sea ice model. This calculation requires an external SOM forcing data file that includes ocean mixed layer depths and bottom-of-the-slab Q-fluxes. Scientifically appropriate bottom-of-the-slab Q-fluxes are normally ocean resolution dependent and are derived from the ocean model output of a fully coupled run. Note that while this mode runs out of the box, the default SOM forcing file is not scientifically appropriate and is provided for testing and development purposes only. Users must create scientifically appropriate data for their particular application. A tool is available to derive valid SOM forcing.

DOCN can be user-customized in three ways.

You can set DOCN run-time variables[40].

You can edit `user_nl_docn` to change namelist settings by adding all user specific namelist changes in the form of "namelist_var = new_namelist_value". Note that any namelist variable from shr_strdata_nml and datm_nml can be modified below using the this syntax. Use **preview_namelists** to view (not modify) the output namelist in `CaseDocs`.

You can modify the contents of a DOCN stream txt file. To do this:

- use **preview_namelists** to obtain the contents of the stream txt files in `CaseDocs/`
- place a *copy* of this file in $CASEROOT with the string *"user_"* prepended
- **Make sure you change the permissions of the file to be writeable** (chmod 644)
- Modify the `user_docn.streams.txt.*` file.

As an example, if the stream text file in `CaseDocs/` is `doc.stream.txt.prescribed`, the modified copy in $CASEROOT should be `user_docn.streams.txt.prescribed`. After changing this file and calling **preview_namelists** again, you should see your new modifications appear in `CaseDocs/docn.streams.txt.prescribed`.

DICE

DICE is discussed in detail in Data Model's User's Guide[41].

DICE can be user-customized in three ways.

You can set DICE run-time variables[42].

You can edit `user_nl_dice` to change namelist settings by adding all user specific namelist changes in the form of "namelist_var = new_namelist_value". Note that any namelist variable from shr_strdata_nml and datm_nml can be modified below using the this syntax. Use **preview_namelists** to view (not modify) the output namelist in `CaseDocs/`.

You can modify the contents of a DICE stream txt file. To do this:

- use **preview_namelists** to obtain the contents of the stream txt files in `CaseDocs/`
- place a *copy* of this file in $CASEROOT with the string *"user_"* prepended
- **Make sure you change the permissions of the file to be writeable** (chmod 644)

- Modify the `user_dice.streams.txt.*` file.

DLND

DLND is discussed in detail in Data Model's User's Guide[43]. The data land model is different from the other data models because it can run as a purely data-land model (reading in coupler history data for atm/land fluxes and land albedos produced by a previous run), or to read in model output from CLM to send to CISM.

DLND can be user-customized in three ways:

You can set DLND run-time variables[44].

You can edit `user_nl_dlnd` OR `user_nl_dsno` depending on the component set, to change namelist settings namelists settings by adding all user specific namelist changes in the form of "namelist_var = new_namelist_value". Note that any namelist variable from shr_strdata_nml and dlnd_nml or dsno_nml can be modified below using the this syntax. Use **preview_namelists** to view (not modify) the output namelist in `CaseDocs/`.

You can modify the contents of a DLND stream txt file. To do this:

- use **preview_namelists** to obtain the contents of the stream txt files in `CaseDocs/`
- place a *copy* of this file in $CASEROOT with the string *"user_"* prepended
- **Make sure you change the permissions of the file to be writeable** (chmod 644)
- Modify the `user_dlnd.streams.txt.*` file.

DROF

DROF is discussed in Data Model's User's Guide[45]. The data river runoff model reads in runoff data and sends it back to the coupler. In general, the data river runoff model is only used to provide runoff forcing data to POP2 when running C or G compsets

DROF can be user-customized in three ways:

You can set DROF run-time variables[46].

You can edit `user_nl_drof` to change namelist settings namelists settings by adding all user specific namelist changes in the form of "namelist_var = new_namelist_value". Note that any namelist variable from shr_strdata_nml and drof_nml can be modified using the this syntax. Use **preview_namelists** to view (not modify) the output namelist in `CaseDocs/`.

You can modify the contents of a DROF stream txt file. To do this:

- use **preview_namelists** to obtain the contents of the stream txt files in `CaseDocs/`
- place a *copy* of this file in $CASEROOT with the string *"user_"* prepended
- **Make sure you change the permissions of the file to be writeable** (chmod 644)
- Modify the `user_drof.streams.txt.*` file.

Controlling output data

During a model run, each CESM component produces its own output datasets consisting of history, restart and output log files. Component history files and restart files are in netCDF format. Restart files are used to either exactly restart the model or to serve as initial conditions for other model cases.

Archiving is a phase of a CESM model run where the generated output data is moved from $RUNDIR to a local disk area (short-term archiving) and subsequently to a long-term storage system (long-term archiving). It has no impact on the production run except to clean up disk space and help manage user quotas. Although short-term and long-term archiving are implemented independently in the scripts, there is a dependence between the two since the short-term archiver must be turned on in order for the long-term archiver to be activated. In `env_run.xml`, several variables control the behavior of short and long-term archiving. See short and long term archiving[47] for a description of output data control variables. Several important points need to be made about both short and long term archiving:

- By default, short-term archiving is enabled and long-term archiving is disabled.
- All output data is initially written to $RUNDIR.
- Unless a user explicitly turns off short-term archiving, files will be moved to $DOUT_S_ROOT at the end of a successful model run.
- If long-term archiving is enabled, files will be moved to $DOUT_L_MSROOT by **$CASE.l_archive**, which is run as a separate batch job after the successful completion of a model run.
- Users should generally turn off short term-archiving when developing new CESM code.
- If long-term archiving is not enabled, users must monitor quotas and usage in the $DOUT_S_ROOT/ directory and should manually clean up these areas on a frequent basis.

Standard output generated from each CESM component is saved in a "log file" for each component in $RUNDIR. Each time the model is run, a single coordinated datestamp is incorporated in the filenames of all output log files associated with that run. This common datestamp is generated by the run script and is of the form YYMMDD-hhmmss, where YYMMDD are the Year, Month, Day and hhmmss are the hour, minute and second that the run began (e.g. ocn.log.040526-082714). Log files are also copied to a user specified directory using the variable $LOGDIR in `env_run.xml`. The default is a 'logs' subdirectory beneath the case directory.

By default, each component also periodically writes history files (usually monthly) in netCDF format and also writes netCDF or binary restart files in the $RUNDIR directory. The history and log files are controlled independently by each component. History output control (i.e. output fields and frequency) is set in the Buildconf/$component.buildnml.csh files.

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

As an example, for the following example settings

```
DOUT_S = TRUE
DOUT_S_ROOT = /ptmp/$user/archive
DOUT_L_MS = TRUE
DOUT_L_MSROOT /USER/csm/$CASE
```

the run will automatically submit the **$CASE.l_archive** to the queue upon its completion to archive the data. The system is not bulletproof, and you will want to verify at regular intervals that the archived data is complete, particularly during long running jobs.

Load balancing a case

Load balancing refers to the optimization of the processor layout for a given model configuration (compset, grid, etc) such that the cost and throughput will be optimal. Optimal is a somewhat subjective thing. For a fixed total number of processors, it means achieving the maximum throughput. For a given configuration across varied processor counts, it means finding several "sweet spots" where the model is minimally idle, the cost is relatively low, and the throughput is relatively high. As with most models, increasing total processors normally results in both increased throughput and increased cost. If models scaled linearly, the cost would remain constant across different processor counts, but generally, models don't scale linearly and cost increases with increasing processor count. This is certainly true for CESM. It is strongly recommended that a user perform a load-balancing exercise on their proposed model run before undertaking a long production run.

CESM has significant flexibility with respect to the layout of components across different hardware processors. In general, there are seven unique models (atm, lnd, rof, ocn, ice, glc, cpl) that are managed independently in CESM, each with a unique MPI communicator. In addition, the driver runs on the union of all processors and controls the sequencing and hardware partitioning.

Please see the section on setting the case PE layout for a detailed discussion of how to set processor layouts and the example on changing the PE layout .

Model timing data

In order to perform a load balancing exercise, you must first be aware of the different types of timing information produced by every CESM run. How this information is used is described in detail in using model timing data.

A summary timing output file is produced after every CESM run. This file is placed in `$CASEROOT/timing/ccsm_timing.$CASE.$date`, where $date is a datestamp set by CESM at runtime, and contains a summary of various information. The following provides a description of the most important parts of a timing file.

The first section in the timing output, CCSM TIMING PROFILE, summarizes general timing information for the run. The total run time and cost is given in several metrics including pe-hrs per simulated year (cost), simulated years per wall day (thoughput), seconds, and seconds per model day. This provides general summary information quickly in several units for analysis and comparison with other runs. The total run time for each component is also provided, as is the time for initialization of the model. These times are the aggregate over the total run and do not take into account any temporal or processor load imbalances.

The second section in the timing output, "DRIVER TIMING FLOWCHART", provides timing information for the driver in sequential order and indicates which processors are involved in the cost. Finally, the timings for the coupler are broken out at the bottom of the timing output file.

Separately, there is another file in the timing directory, ccsm_timing_stats.$date that accompanies the above timing summary. This second file provides a summary of the minimum and maximum of all the model timers.

There is one other stream of useful timing information in the cpl.log.$date file that is produced for every run. The cpl.log file contains the run time for each model day during the model run. This diagnostic is output as the model runs. You can search

for tStamp in the cpl.log file to see this information. This timing information is useful for tracking down temporal variability in model cost either due to inherent model variability cost (I/O, spin-up, seasonal, etc) or possibly due to variability due to hardware. The model daily cost is generally pretty constant unless I/O is written intermittently such as at the end of the month.

Using model timing data

In practice, load-balancing requires a number of considerations such as which components are run, their absolute and relative resolution; cost, scaling and processor count sweet-spots for each component; and internal load imbalance within a component. It is often best to load balance the system with all significant run-time I/O turned off because this occurs very infrequently, typically one timestep per month, and is best treated as a separate cost as it can bias interpretation of the overall model load balance. Also, the use of OpenMP threading in some or all of the components is dependent on the hardware/OS support as well as whether the system supports running all MPI and mixed MPI/OpenMP on overlapping processors for different components. A final point is deciding whether components should run sequentially, concurrently, or some combination of the two with each other. Typically, a series of short test runs is done with the desired production configuration to establish a reasonable load balance setup for the production job. The timing output can be used to compare test runs to help determine the optimal load balance.

Changing the pe layout of the model has NO IMPACT on the scientific results. The basic order of operations and calling sequence is hardwired into the driver and that doesn't change when the pe layout is changed. There are some constraints on the ability of CESM to run fully concurrent. In particular, the atmosphere model always run sequentially with the ice and land for scientific reasons. As a result, running the atmosphere concurrently with the ice and land will result in idle processors in these components at some point in the timestepping sequence. For more information about how the driver is implemented, see (Craig, A.P., Vertenstein, M., Jacob, R., 2012: A new flexible coupler for earth system modeling developed for CCSM4 and CESM1.0. International Journal of High Performance Computing Applications, 26, 31-42, 10.1177/1094342011428141). As of CESM1.1.1, there is a new separate rof component. That component is implemented in the driver just like the land model. It can run concurrently with the land model but not concurrently with the atmosphere model.

In general, we normally carry out 20-day model runs with restarts and history turned off in order to find the layout that has the best load balance for the targeted number of processors. This provides a reasonable performance estimate for the production run for most of the runtime. The end of month history and end of run restart I/O is treated as a separate cost from the load balance perspective. To set up this test configuration, create your production case, and then edit env_run.xml and set STOP_OPTION to ndays, STOP_N to 20, and RESTART_OPTION to never. Seasonal variation and spin-up costs can change performance over time, so even after a production run has started, it's worthwhile to occasionally review the timing output to see whether any changes might be made to the layout to improve throughput or decrease cost.

In determining an optimal load balance for a specific configuration, two pieces of information are useful.

- Determine which component or components are most expensive.
- Understand the scaling of the individual components, whether they run faster with all MPI or mixed MPI/OpenMP decomposition strategies, and their optimal decompositions at each processor count. If the cost and scaling of the components are unknown, several short tests can be carried out with arbitrary component pe counts just to establish component scaling and sweet spots.

One method for determining an optimal load balance is as follows

- start with the most expensive component and a fixed optimal processor count and decomposition for that component
- test the systems, varying the sequencing/concurrency of the components and the pe counts of the other components
- identify a few best potential load balance configurations and then run each a few times to establish run-to-run variability and to try to statistically establish the faster layout

In all cases, the component run times in the timing output file can be reviewed for both overall throughput and independent component timings. Using the timing output, idle processors can be identified by considering the component concurrency in conjunction with the component timing.

In general, there are only a few reasonable component layout options for CESM.

- fully sequential
- fully sequential except the ocean running concurrently
- fully concurrent except the atmosphere run sequentially with the ice, rof, and land components
- finally, it makes best sense for the coupler to run on a subset of the atmosphere processors and that can be sequentially or concurrently run with the land and ice

The concurrency is limited in part by the hardwired sequencing in the driver. This sequencing is set by scientific constraints, although there may be some addition flexibility with respect to concurrency when running with mixed active and data models.

There are some general rules for finding optimal configurations:

- Make sure you have set a processor layout where each hardware processor is assigned to at least one component. There is rarely a reason to have completely idle processors in your layout.
- Make sure your cheapest components keep up with your most expensive components. In other words, a component that runs on 1024 processors should not be waiting on a component running on 16 processors.
- Before running the job, make sure the batch queue settings in the `$CASE.run` script are set correctly for the specific run being targetted. The account numbers, queue names, time limits should be reviewed. The ideal time limit, queues, and run length are all dependent on each other and on the current model throughput.
- Make sure you are taking full advantage of the hardware resources. If you are charged by the 32-way node, you might as well target a total processor count that is a multiple of 32.
- If possible, keep a single component on a single node. That usually minimizes internal component communication cost. That's obviously not possible if running on more processors than the size of a node.
- And always assume the hardware performance could have variations due to contention on the interconnect, file systems, or other areas. If unsure of a timing result, run cases multiple times.

How do I run a case?

Setting the time limits

Before you can run the job, you need to make sure the batch queue variables are

set correctly for the specific run being targeted. This is done currently by manually editing `$CASE.run`. You should carefully check the batch queue submission lines and make sure that you have appropriate account numbers, time limits, and stdout file names. In looking at the ccsm_timing.$CASE.$datestamp files for "Model Throughput", output like the following will be found:

```
Overall Metrics:
Model Cost: 327.14 pe-hrs/simulated_year (scale= 0.50)
Model Throughput: 4.70 simulated_years/day
```

The model throughput is the estimated number of model years that you can run in a wallclock day. Based on this, you can maximize $CASE.run queue limit and change $STOP_OPTION and $STOP_N in `env_run.xml`. For example, say a model's throughput is 4.7 simulated_years/day. On yellowstone(??), the maximum runtime limit is 6 hours. 4.7 model years/24 hours * 6 hours = 1.17 years. On the massively parallel computers, there is always some variability in how long it will take a job to run. On some machines, you may need to leave as much as 20% buffer time in your run to guarantee that jobs finish reliably before the time limit. For that reason we will set our model to run only one model year/job. Continuing to assume that the run is on yellowstone, in `$CASE.yellowstone.run` set

```
#BSUB -W 6:00
```

and `xmlchange` should be invoked as follows in CASEROOT:

```
./xmlchange STOP_OPTION=nyears
./xmlchange STOP_N=1
./xmlchange REST_OPTION=nyears
./xmlchange REST_N=1
```

Submitting the run

Once you have configured and built the model, submit `$CASE.run` to your machine's batch queue system using the `$CASE.submit` command.

```
> $CASE.submit
```

You can see a complete example of how to run a case in the basic example.

When executed, the run script, `$CASE.run`:

- Will not execute the build script. Building CESM is now done only via an interactive call to the build script, **$CASE.build**.
- Will check that locked files are consistent with the current xml files, run the buildnml script for each component and verify that required input data is present on local disk (in $DIN_LOC_ROOT).
- Will run the CESM model.
- Upon completion, will put timing information in $LOGDIR/timing and copy log files back to $LOGDIR
- If $DOUT_S is TRUE, component history, log, diagnostic, and restart files will be moved from $RUNDIR to the short-term archive directory, $DOUT_S_ROOT.
- If $DOUT_L_MS is TRUE, the long-term archiver, **$CASE.l_archive**, will be submitted to the batch queue upon successful completion of the run.
- If $RESUBMIT >0, resubmit **$CASE.run**

If the job runs to completion, you should have "SUCCESSFUL TERMINATION OF CPL7-CCSM" near the end of your STDOUT file. New data should be in the subdi-

rectories under $DOUT_S_ROOT, or if you have long-term archiving turned on, it should be automatically moved to subdirectories under $DOUT_L_MSROOT.

If the job failed, there are several places where you should look for information. Start with the STDOUT and STDERR file(s) in $CASEROOT. If you don't find an obvious error message there, the $RUNDIR/$model.log.$datestamp files will probably give you a hint. First check cpl.log.$datestamp, because it will often tell you when the model failed. Then check the rest of the component log files. Please see troubleshooting runtime errors for more information.

REMINDER: Once you have a successful first run, you must set CONTINUE_RUN to TRUE in `env_run.xml` before resubmitting, otherwise the job will not progress. You may also need to modify the STOP_OPTION, STOP_N and/or STOP_DATE[48], REST_OPTION, REST_N and/or REST_DATE[49], and RESUBMIT[50] variables in `env_run.xml` before resubmitting.

Restarting a run

Restart files are written by each active component (and some data components) at intervals dictated by the driver via the setting of the `env_run.xml` variables, $REST_OPTION and $REST_N. Restart files allow the model to stop and then start again with bit-for-bit exact capability (i.e. the model output is exactly the same as if it had never been stopped). The driver coordinates the writing of restart files as well as the time evolution of the model. All components receive restart and stop information from the driver and write restarts or stop as specified by the driver.

It is important to note that runs that are initialized as branch or hybrid runs, will require restart/initial files from previous model runs (as specified by the variables, $RUN_REFCASE and $RUN_REFDATE). These required files must be prestaged *by the user* to the case $RUNDIR (normally `$EXEROOT/run`) before the model run starts. This is normally done by just copying the contents of the relevant `$RUN_REFCASE/rest/$RUN_REFDATE.00000` directory.

Whenever a component writes a restart file, it also writes a restart pointer file of the form, `rpointer.$component`. The restart pointer file contains the restart filename that was just written by the component. Upon a restart, each component reads its restart pointer file to determine the filename(s) to read in order to continue the model run. As examples, the following pointer files will be created for a component set using full active model components.

- rpointer.atm
- rpointer.drv
- rpointer.ice
- rpointer.lnd
- rpointer.rof
- rpointer.cism
- rpointer.ocn.ovf
- rpointer.ocn.restart

If short-term archiving is turned on, then the model archives the component restart datasets and pointer files into `$DOUT_S_ROOT/rest/yyyy-mm-dd-sssss`, where yyyy-mm-dd-sssss is the model date at the time of the restart (see below for more details). If long-term archiving these restart then archived in `$DOUT_L_MSROOT/rest`. DOUT_S_ROOT and DOUT_L_MSROOT are set in `env_run.xml`, and can be changed at any time during the run.

Backing up to a previous restart

If a run encounters problems and crashes, you will normally have to back up to a previous restart. Assuming that short-term archiving is enabled, you will need to find the latest `$DOUT_S_ROOT/rest/yyyy-mm-dd-ssss/` directory that was created and copy the contents of that directory into your run directory ($RUNDIR). You can then continue the run and these restarts will be used. It is important to make sure the new rpointer.* files overwrite the rpointer.* files that were in $RUNDIR, or the job may not restart in the correct place.

Occasionally, when a run has problems restarting, it is because the rpointer files are out of sync with the restart files. The rpointer files are text files and can easily be edited to match the correct dates of the restart and history files. All the restart files should have the same date.

Data flow during a model run

All component log files are copied to the directory specified by the `env_run.xml` variable $LOGDIR which by default is set to `$CASEROOT/logs`. This location is where log files are copied when the job completes successfully. If the job aborts, the log files will NOT be copied out of the `$RUNDIR` directory.

Once a model run has completed successfully, the output data flow will depend on whether or not short-term archiving is enabled (as set by the `env_run.xml` variable, $DOUT_S). By default, short-term archiving will be done.

No archiving

If no short-term archiving is performed, then all model output data will remain in the run directory, as specified by the `env_run.xml` variable, $RUNDIR. Furthermore, if short-term archiving is disabled, then long-term archiving will not be allowed.

Short-term archiving

If short-term archiving is enabled, the component output files will be moved to the short term archiving area on local disk, as specified by $DOUT_S_ROOT. The directory DOUT_S_ROOT is normally set to $EXEROOT/../archive/$CASE. and will contain the following directory structure:

```
atm/
    hist/ logs/
cpl/
    hist/ logs/
glc/
    logs/
ice/
    hist/ logs/
lnd/
    hist/ logs/
ocn/
    hist/ logs/
rest/
    yyyy-mm-dd-sssss/
    ....
    yyyy-mm-dd-sssss/
```

`hist/` contains component history output for the run.

`logs/` contains component log files created during the run. In addition to $LOGDIR, log files are also copied to the short-term archiving directory and therefore are available for long-term archiving.

`rest/` contains a subset of directories that each contain a *consistent* set of restart files, initial files and rpointer files. Each sub-directory has a unique name corresponding to the model year, month, day and seconds into the day where the files were created (e.g. `1852-01-01-00000/`). The contents of any restart directory can be used to create a branch run or a hybrid run or back up to a previous restart date.

Long-term archiving

For long production runs that generate many giga-bytes of data, you will normally want to move the output data from local disk to a long-term archival location. Long-term archiving can be activated by setting $DOUT_L_MS to TRUE in `env_run.xml`. By default, the value of this variable is FALSE, and long-term archiving is disabled. If the value is set to TRUE, then the following additional variables are: $DOUT_L_MSROOT, $DOUT_S_ROOT DOUT_S (see variables for output data management).

As was mentioned above, if long-term archiving is enabled, files will be moved out of $DOUT_S_ROOT to $DOUT_L_ROOT by `$CASE.l_archive`„ which is run as a separate batch job after the successful completion of a model run.

Testing a case

After the case has built and has demonstrated the ability to run via a short test, it is important to formally test exact restart capability before a production run is started. See the Section called *Using* ***create_production_test*** in Chapter 7 for more information about how to use create_production_test.

Notes

1. ../modelnl/env_run.html
2. ../modelnl/env_run.html#run_start
3. ../modelnl/env_run.html#run_stop
4. ../modelnl/env_run.html#run_rest
5. ../modelnl/env_run.html
6. ../modelnl/env_build.html
7. ../modelnl/modelnl.html
8. ../modelnl/nl_drv.html
9. ../modelnl/env_case.html
10. ../modelnl/env_mach_pes.html
11. ../modelnl/env_run.html
12. http://www.cesm.ucar.edu/models/cesm1.1/cam/docs/ug5_2/book1.html
13. http://www.cesm.ucar.edu/models/cesm1.1/cam/docs/ug5_2/book1.html
14. ../modelnl/env_build.html#build_cam
15. ../modelnl/env_run.html#run_cam
16. ../modelnl/env_run.html#run_cam

17. ../modelnl/nl_cam.html
18. http://www.cesm.ucar.edu/models/cesm1.0/clm/models/lnd/clm/doc/UsersGuide/book1.html
19. http://www.cesm.ucar.edu/models/cesm1.0/clm/models/lnd/clm/doc/UsersGuide/book1.html
20. ../modelnl/env_build.html#build_clm
21. ../modelnl/env_run.html#run_clm
22. ../modelnl/nl_clm.html
23. ../modelnl/nl_rtm.html
24. http://www.cesm.ucar.edu/models/cesm1.1/cice/doc/index.html
25. http://www.cesm.ucar.edu/models/cesm1.1/cice/doc/index.html
26. ../modelnl/env_build.html#build_cice
27. ../modelnl/env_run.html#run_cice
28. ../modelnl/nl_cice.html
29. ../modelnl/env_build.html#build_cice
30. ../modelnl/nl_pop2.html
31. ../modelnl/env_run.html#run_pop
32. ../modelnl/nl_cice.html
33. ../modelnl/env_build.html#build_pop2
34. ../modelnl/nl_cism.html
35. ../modelnl/env_run.html#run_cism
36. http://www.cesm.ucar.edu/models/cesm1.1/data8/doc/book1.html
37. ../modelnl/env_run.html#run_datm
38. http://www.cesm.ucar.edu/models/cesm1.1/data8/doc/book1.html
39. ../modelnl/env_run.html#run_sstice
40. ../modelnl/env_run.html#run_docn
41. http://www.cesm.ucar.edu/models/cesm1.1/data8/doc/book1.html
42. ../modelnl/env_run.html#run_dice
43. http://www.cesm.ucar.edu/models/cesm1.1/data8/doc/book1.html
44. ../modelnl/env_run.html#run_dlnd
45. http://www.cesm.ucar.edu/models/cesm1.1/data8/doc/book1.html
46. ../modelnl/env_run.html#run_drof
47. ../modelnl/env_run.html#run_datout
48. ../modelnl/env_run.html#run_stop
49. ../modelnl/env_run.html#run_rest
50. ../modelnl/env_run.html#run_rest

Chapter 5. Porting and Validating CESM on a new platform

Porting Overview

One of the first steps many users will have to address is getting the CESM model running on their local machine. This section will describe the process of going about that. In short, you should first call **create_newcase** using a "userdefined" machine name and get that case running. Second, you should take the results of the previous step and introduce your machine in the `$CCSMROOT/scripts/ccsm_utils/Machines/` directory so that your local machine is supported out-of-the-box. This greatly eases setting up cases and benefits groups of users by requiring the port be done only once. Third you should validate the model on your machine.

It is usually very helpful to assure that you can run a basic mpi parallel program on your machine prior to attempting a CESM port. Understanding how to compile and run the program fhello_world_mpi.F90 shown here could potentially save many hours of frustration.

```
program fhello_world_mpi.F90
  use mpi
  implicit none
  integer ( kind = 4 ) error
  integer ( kind = 4 ) id
  integer p
  character(len=MPI_MAX_PROCESSOR_NAME) :: name
  integer clen
  integer, allocatable :: mype(:)
  real ( kind = 8 ) wtime

  call MPI_Init ( error )
  call MPI_Comm_size ( MPI_COMM_WORLD, p, error )
  call MPI_Comm_rank ( MPI_COMM_WORLD, id, error )
  if ( id == 0 ) then

    wtime = MPI_Wtime ( )

    write ( *, '(a)' ) ' '
    write ( *, '(a)' ) 'HELLO_MPI - Master process:'
    write ( *, '(a)' ) '  FORTRAN90/MPI version'
    write ( *, '(a)' ) ' '
    write ( *, '(a)' ) '  An MPI test program.'
    write ( *, '(a)' ) ' '
    write ( *, '(a,i8)' ) '  The number of processes is ', p
    write ( *, '(a)' ) ' '

  end if

  call MPI_GET_PROCESSOR_NAME(NAME, CLEN, ERROR)

  write ( *, '(a)' ) ' '
  write ( *, '(a,i8,a,a)' ) '  Process ', id, ' says "Hello, world!" ',name(1:clen)

  call MPI_Finalize ( error )

end program
```

You will want to start with an X (i.e. commonly referred to as dead) compset running at a low resolution. So you could, for instance, start with an X compset at resolution f45_g37. This will allow you to determine whether all prerequisite software is in place and working for a simple parallel CESM configuration that requires minimal input data. Once that is working move to an A compset with resolution f45_g37. Once that's working, run a B compset at resolution f45_g37. Finally when all the previous steps have run correctly, run your target compset and resolution.

Step 1: Use create_newcase with a userdefined machine name

This section describes how to set up a case using a userdefined machine name and then within that case, how to modify the scripts to get that case running on a local machine.

1. Run **create_newcase** wtih a "userdefined" machine name. Then run **cesm_setup** in the new case directory.

```
> cd $CCSMROOT/scripts
> create_newcase -case test1 \
                 -res f45_g37 \
                 -compset X \
                 -mach userdefined
> cd test1
> cesm_setup
```

   The output from **cesm_setup** will indicate which xml variables you are now required to set.

```
ERROR: must set xml variable OS to generate Macros file
ERROR: must set xml variable MAX_TASKS_PER_NODE to build the model
ERROR: must set xml variable MPILIB to build the model
ERROR: must set xml variable RUNDIR to build the model
ERROR: must set xml variable DIN_LOC_ROOT to build the model
ERROR: must set xml variable COMPILER to build the model
ERROR: must set xml variable EXEROOT to build the model
Correct above and issue cesm_setup again
```

   The definition of every env variable can be found on the CASEROOT xml page[1]. Enter appropriate settings for the above xml variables in `env_build.xml`, `env_mach_pes.xml` and `env_run.xml`. Calling **cesm_setup** again should now produce a `Macros` file that can be used as a starting point for your port. In addition build and run scripts will be generated.

2. The next step is to edit the `env_mach_specific` and `Macros` files to get ready to build the model. The string USERDEFINED in these files indicate the locations where modifications are likely. In particular `env_mach_specific` is where modules, paths, or machine environment variables need to be set especially related to compilers, mpi, and netcdf. `Macros` is where the Makefile variables are set. You can find the `Makefile` in the Tools directory. In the `Macros`, modify SLIBS to include whatever machine specific libs are desired and include the netcdf library or libraries. Then set NETCDF_PATH to the path of the netcdf directory. This might be a hardwired path or it might be an env variable set in `env_mach_specfic` or through modules. You might need to modify other Macros variables such as MPI_PATH, but that depends on your particular system setup. Often mpi is wrapped in the compiler commands like mpif90 automatically.

   As an example, suppose your machine uses Modules (i.e. the Modules package provides for the dynamic modification of a user's environment via modulefiles). The following setting from env_mach_specific.bluewaters sets the compiler and netcdf versions.

```
# invoking modules sets $MPICH_DIR and $NETCDF_DIR
if ( $COMPILER == "pgi" ) then
    module load PrgEnv-pgi
    module switch pgi       pgi/11.10.0
endif
module load torque/2.5.10
module load netcdf-hdf5parallel/4.1.3
module load parallel-netcdf/1.2.0
```

   that produces some env variables which can then be used in the generated Macros as follows:

```
MPI_PATH:= $(MPICH_DIR)
```

```
NETCDF_PATH:= $(NETCDF_DIR)
```

So in this example the system module defines a variable NETCDF_DIR, but CESM expects NETCDF_PATH to be set and that copy is made in the Macros file. While CESM supports use of pnetcdf in PIO (which requires setting PNETCDF_PATH in Macros), it is generally best to ignore that feature during initial porting. PIO works well with standard NetCDF.

3. Build the case

```
> ./test1.userdefined.build
```

This step will often fail if paths to compilers, compiler versions, or libraries are not set properly, if compiler options are not set properly, or if machine environment variables are not set properly. Review and edit the `env_mach_specific` and `Macros` files, clean the build,

```
> ./test1.userdefined.clean_build
```

and try rebuilding again.

4. Finally `/test1.userdefined.run` is the job submission or run script. Modifications are needed to specify the local batch environment and the job launch command. Again, the string USERDEFINED will indicate where those changes are needed. Once the batch and launch commands are set, run the model using your local job submission command. qsub is used here for example.

```
> qsub test1.userdefined.run
```

The job will fail to submit if the batch commands are not set properly. The job could fail to run if the launch command is incorrect or if the batch commands are not set consistent with the job resource needs. Review the run script and try resubmitting.

Step 2: Enabling out-of-the box capability for your machine

Once a case is running, then the local setup for the case can be converted into a specific set of machine files, so future cases can be set up using your local machine name rather than "userdefined". In addition, new cases should be able to run out-of-the-box without going through step 1 above. Basically, you will need to add files and modify files in the directory `$CCSMROOT/scripts/ccsm_utils/Machines` to support your machine out-of-the-box. This section describes how to add support for your machine to the CESM scripts in order to support your machine out-of-the box.

1. Pick a name that will be associated with your machine. Generally, this will be identical to the name of your machine, but it could be anything. "wilycoyote" will be used in the description to follow. It is also helpful to identify as a starting point one or more supported machines[2] that are similar to your machine. To add wilycoyote to the list of supported machines, do the following:
2. Edit `config_machines.xml` and add a section for "wilycoyote". You can simply copy one of the existing entries and then edit it. The machine specific env variables that need to be set in config_machines.xml for wilycoyote are already set in the env files in the test1 case directory that was created from the userdefined machine. You will need to leverage the variables you used in the test1 case directory in Step1 above into the `config_machines.xml` section for wilycoyote. While the compiler options for a given compiler are pretty consistent across machines, invoking the compiler and the local paths for libraries are not. There are several variable settings here. The definition of these variables can be found in the env_build.xml[3], env_run.xml[4] and env_mach_pes.xml[5] files. Some of the important ones are MACH which should be set to wilycoyote, EXEROOT which should be set to a generic working directory like /tmp/scratch/$CCSMUSER/$CASE shared by and write accessable to all compute nodes, DIN_LOC_ROOT which should be set

to the path to the CESM inputdata directory (read accessable to all compute nodes), BATCHQUERY and BATCHJOBS which specify the query and submit command lines for batch jobs and are used to chain jobs together in production, and MAX_TASKS_PER_NODE which set the maximum number of tasks allowed on each hardware node.

3. Edit `config_compilers.xml` to translate the additions you made to the Macros file to support "wilycoyote" specific settings.
4. Create an `env_mach_specific.wilycoyote` file. This should be a copy of the env_mach_specific file from the test1 case directory in Step1 above.

```
> cd $CCSMROOT/scripts/test1
> cp env_mach_specific  $CCSMROOT/scripts/ccsm_utils/Machines/env_mach_specific.wi
```

5. Create an `mkbatch.wilycoyote` file. The easiest way to do this is to find a machine closest to your machine and copy that file to `mkbatch.wilycoyote`. Then edit `mkbatch.wilycoyote` to match the changes made in the test1.userdefined.run file in the test1 case in Step1. In particular, the batch commands and the job launching will probably need to be changed. The batch commands and setup are the first section of the script. The job launching can be found by searching for the string "CSM EXECUTION".
6. Test the new machine setup. Create a new case based on test1 using the wilycoyote machine setup

```
> cd $CCSMROOT/scripts
> create_newcase -case test1_wilycoyote \
                 -res f45_g37 \
                 -compset X \
                 -mach wilycoyote
> cd test1_wilycoyote
> ./cesm_setup
> ./test1_wilycoyote.build
> qsub test1_wilycoyote.run
```

The point is to confirm that test1_wilycoyote runs fine and is consistent with the original test1 case. Once that works, test other configurations then move to port validation, see the Section called *Step 3: Port Validation*. You should expect that getting this to work will be an iterative process. Changes will probably be made in both the `config_machines.xml` and in `config_compilers.xml>` Whenever either of these machine files are updated, a new case should be set up. Whenever something is changed in the case scripts to fix a problem, that change should be migrated back to the wilycoyote settings in the machine files. Once a case is running, those changes in the case need to be backed out into the wilycoyote machine files and then those machine files can be tested with a new case. Eventually, the machine files should work for any user and any configuration for wilycoyote.

Step 3: Port Validation

The following port validation is recommended for any new machine. Carrying out these steps does not guarantee the model is running properly in all cases nor that the model is scientifically valid on the new machine. In addition to these tests, detailed validation should be carried out for any new production run. That means verifying that model restarts are bit-for-bit identical with a baseline run, that the model is bit-for-bit reproducible when identical cases are run for several months, and that production cases are monitored very carefully as they integrate forward to identify any potential problems as early as possible. These are recommended steps for validating a port and are largely functional tests. Users are responsible for their own validation process, especially with respect to science validation.

1. Verify functionality by performing these functionality tests.

```
ERS_D.f19_g16.X
ERS_D.T31_g37.A
ERS_D.f19_g16.B1850CN
ERI.ne30_g16.X
ERI.T31_g37.A
ERI.f19_g16.B1850CN
ERS.ne30_ne30.F
ERS.f19_g16.I
ERS.T62_g16.C
ERS.T62_g16.DTEST
ERT.ne30_g16.B1850CN
```

2. Verify performance and scaling analysis.
    a. Create one or two load-balanced configurations to check into `Machines/config_pes.xml` for the new machine.
    b. Verify that performance and scaling are reasonable.
    c. Review timing summaries in $CASEROOT for load balance and throughput.
    d. Review coupler "daily" timing output for timing inconsistencies. As has been mentioned in the section on load balancing a case , useful timing information is contained in cpl.log.$date file that is produced for every run. The cpl.log file contains the run time for each model day during the model run. This diagnostic is output as the model runs. You can search for tStamp in this file to see this information. This timing information is useful for tracking down temporal variability in model cost either due to inherent model variability cost (I/O, spin-up, seasonal, etc) or possibly due to variability due to hardware. The model daily cost is generally pretty constant unless I/O is written intermittently such as at the end of the month.

3. Perform validation (both functional and scientific):
    a. Perform a CAM error growth test[6].
    b. Follow the CCSM4.0 CICE port-validation procedure.[7]
    c. Follow the CCSM4.0 POP2 port-validation procedure.[8]

4. Perform two, one-year runs (using the expected load-balanced configuration) as separate job submissions and verify that atmosphere history files are bfb for the last month. Do this after some performance testing is complete; you may also combine this with the production test by running the first year as a single run and the second year as a multi-submission production run. This will test reproducibility, exact restart over the one-year timescale, and production capability all in one test.
5. Carry out a 20-30 year 1.9x2.5_gx1v6 resolution, B_1850_CN compset simulation and compare the results with the diagnostics plots for the 1.9x2.5_gx1v6 Pre-Industrial Control (see the CCSM4.0 diagnostics [9]). Model output data for these runs will be available on the Earth System Grid (ESG) [10] as well.

Notes

1. ../modelnl
2. ../modelnl/machines.html
3. ../modelnl/env_build.html
4. ../modelnl/env_run.html
5. ../modelnl/env_mach_pes.html

6. http://www.cesm.ucar.edu/models/cesm1.1/cam/docs/port/
7. http://www.cesm.ucar.edu/models/cesm1.0/cice/validation/index.html
8. http://www.cesm.ucar.edu/models/cesm1.0/pop2/validation/index.html
9. http://www.cesm.ucar.edu/experiments/cesm1.0/diagnostics/
10. http://www.cesm.ucar.edu/models/cesm1.0/model_esg/

Chapter 6. Use Cases and FAQs

BASICS: A basic example

This specifies all the steps necessary to create, set up, build, and run a case. The following assumes that $CCSMROOT is /user/ccsmroot.

1. Create a new case named EXAMPLE_CASE in the ~/cesm directory. Use an 1850 control compset at 1-degree resolution on yellowstone.

```
> cd /user/ccsmroot/scripts
> ./create_newcase -case ~/cesm/EXAMPLE_CASE \
                   -compset B_1850_CN \
                   -res 0.9x1.25_gx1v6 \
                   -mach yellowstone
```

2. Go to the $CASEROOT directory. Edit `env_mach_pes.xml` if a different pe-layout is desired first. Then set up and build the case.

```
> cd ~/cesm/EXAMPLE_CASE
> ./cesm_setup
> ./EXAMPLE_CASE.build
```

3. Create a production test. Go to the test directory. Build the test first, then run the test and check the TestStatus (the first word should be PASS).

```
> cd ~/cesm/EXAMPLE_CASE
> ./create_production_test
> cd ../EXAMPLE_CASE_ERT
> ./EXAMPLE_CASE_ERT.test_build
> ./EXAMPLE_CASE_ERT.submit
Wait for test to finish.....
> cat TestStatus
```

4. Go back to the case directory, set the job to run 12 model months, use an editor to change the time limit in the run file to accommodate a 12-month run, and submit the job.

```
> cd ../EXAMPLE_CASE
> xmlchange STOP_OPTION=nmonths
> xmlchange STOP_N=12
> # use an editor to change EXAMPLE_CASE.run "#BSUB -W 4:00" to "#BSUB -W 6:00"
> ./EXAMPLE_CASE.submit
```

5. Make sure the run succeeded. Look for the following line at the end of the cpl.log file in your run directory.

```
(seq_mct_drv): ===============       SUCCESSFUL TERMINATION OF CPL7-CCSM =========
```

6. Set it to resubmit itself 10 times so that it will run a total of 11 years (including the initial year), and resubmit the case. (Note that a resubmit will automatically change the run to be a continuation run).

```
> xmlchange RESUBMIT=10
> ./EXAMPLE_CASE.submit
```

BASICS: How do I set up a branch or hybrid run?

The section setting the case initialization discussed starting a new case as a branch run or hybrid run by using data from a previous run. First you need to create a new case. Assume that $CCSMROOT is set to /user/ccsmroot and that $EXEROOT is `/glade/scratch/$user/EXAMPLE_CASEp`. Finally, assume that the branch or hybrid run is being carried out on NCAR's IBM system, yellowstone.

```
> cd /user/ccsmroot/scripts
> create_newcase -case ~/cesm/EXAMPLE_CASEp \
                 -compset B_2000 \
                 -res 0.9x1.25_gx1v6 \
                 -mach yellowstone
> cd ~/cesm/EXAMPLE_CASEp
```

For a branch run, modify `env_run.xml` to branch from EXAMPLE_CASE at year 0001-02-01.

```
> xmlchange RUN_TYPE=branch
> xmlchange RUN_REFCASE=EXAMPLE_CASE
> xmlchange RUN_REFDATE=0001-02-01
```

For a hybrid run, modify `env_run.xml` to start up from EXAMPLE_CASE at year 0001-02-01.

```
> xmlchange RUN_TYPE=hybrid
> xmlchange RUN_REFCASE=EXAMPLE_CASE
> xmlchange RUN_REFDATE=0001-02-01
```

For a branch run, `env_run.xml` for EXAMPLE_CASEp should be identical to EXAMPLE_CASE, except for the $RUN_TYPE setting. In addition, any modifications introduced into any of the `~/cesm/EXAMPLE_CASE/user_nl_*` files, should be reintroduced into the corresponding files in EXAMPLE_CASEp.

Set up and build the case executable.

```
> ./cesm_setup
> ./EXAMPLE_CASEp.build
```

Prestage the necessary restart/initial data in $RUNDIR (assumed to be `/glade/scratch/$user/EXAMPLE_CASEp/run`). Note that `/glade/scratch/$user/EXAMPLE_CASEp/run` was created during the build. Assume that the restart/initial data is on the NCAR HPSS.

```
> cd /glade/scratch/$user/EXAMPLE_CASEp/run
> hsi -q "cget /CCSM/csm/EXAMPLE_CASE/rest/0001-02-01-00000/*"
```

It is assumed that you already have a valid load-balanced scenario. Go back to the case directory, set the job to run 12 model months, use an editor to change the time limit in the run file to accommodate a 12-month run, then submit the job.

```
> cd ~/cesm/EXAMPLE_CASEp
> xmlchange STOP_OPTION=nmonths
> xmlchange STOP_N=12
> # use an editor to change EXAMPLE_CASE.run "#BSUB -W 1:30" to "#BSUB -W 6:00"
> ./EXAMPLE_CASEp.submit
```

Make sure the run succeeded. Look for the following line at the end of the cpl.log file in your run directory.

```
(seq_mct_drv): ===============       SUCCESSFUL TERMINATION OF CPL7-CCSM ==============
```

Change the run to a continuation run. Set it to resubmit itself 10 times so that it will run a total of 11 years (including the initial year), then resubmit the case.

```
> xmlchange CONTINUE_RUN=TRUE
> xmlchange RESUMIT=10
> ./EXAMPLE_CASEp.submit
```

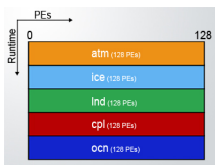
BASICS: What calendars are supported in CESM?

CESM supports a 365 day (or no-leap) calendar as well as a gregorial calendar. The calendar is set by the xml variable, CALENDAR, in env_build.xml[1]. The no-leap calendar has the standard 12 months, but it has 365 days every year and 28 days in every February. Monthly averages in CESM are truly computed over varying number of days depending on the month of the year. In CESM1.0.x, a gregorian calendar was only possible if the ESMF library was used. This is no longer the case in CESM1.1.x and CESM1.2.x.

BASICS: How do I change processor counts and component layouts on processors?

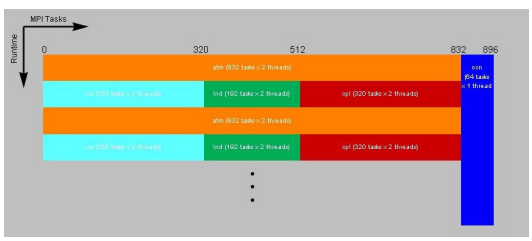
This example modifies the PE layout for our original run, EXAMPLE_CASE. We now target the model to run on the yellowstone supercomputer and modify our PE layout to use a common load balanced configuration for CESM on large IBM machines. Also see the Section called *Changing the PE layout* in Chapter 2.

In our original example, EXAMPLE_CASE, we used 128 pes with each component running sequentially over the entire set of processors.



128-pes/128-tasks layout

Now we change the layout to use 1728 processors and run the ice, lnd, and cpl models concurrently on the same processors as the atm model while the ocean model will run on its own set of processors. The atm model will be run on 1664 pes using 832 MPI tasks each threaded 2 ways and starting on global MPI task 0. The ice model is run using 320 MPI tasks starting on global MPI task 0, but not threaded. The lnd model is run on 384 processors using 192 MPI tasks each threaded 2 ways starting at global MPI task 320 and the coupler is run on 320 processors using 320 MPI tasks starting at global MPI task 512. The ocn model uses 64 MPI tasks starting at global MPI task 832.



1728-pes/896-tasks layout

Since we will be modifying `env_mach_pes.xml` after **cesm_setup** was called, the following needs to be invoked:

```
> ./cesm_setup -clean
> xmlchange NTASKS_ATM=832
> xmlchange NTHRDS_ATM=2
> xmlchange ROOTPE_ATM=0
> xmlchange NTASKS_CPL=320
> xmlchange NTHRDS_CPL=1
> xmlchange ROOTPE_CPL=512
> xmlchange NTASKS_GLC=320
```

```
> xmlchange NTHRDS_GLC=1
> xmlchange ROOTPE_GLC=0
> xmlchange NTASKS_ICE=320
> xmlchange NTHRDS_ICE=1
> xmlchange ROOTPE_ICE=0
> xmlchange NTASKS_LND=192
> xmlchange NTHRDS_LND=2
> xmlchange ROOTPE_LND=320
> xmlchange NTASKS_OCN=64
> xmlchange NTHRDS_OCN=1
> xmlchange ROOTPE_OCN=832
> xmlchange NTASKS_ROF=192
> xmlchange NTHRDS_ROF=2
> xmlchange ROOTPE_ROF=320
> ./cesm_setup
```

It is interesting to compare the timings from the 128- and 1728-processor runs. The timing output below shows that the original model run on 128 pes cost 851 pe-hours/simulated_year. Running on 1728 pes, the model cost more than 5 times as much, but it runs more than two and a half times faster.

```
128-processor case:
Overall Metrics:
Model Cost: 851.05 pe-hrs/simulated_year (scale= 1.00)
Model Throughput: 3.61 simulated_years/day

1728-processor case:
Overall Metrics:
Model Cost: 4439.16 pe-hrs/simulated_year (scale= 1.00)
Model Throughput: 9.34 simulated_years/day
```

See understanding load balancing CESM for detailed information on understanding timing files.

BASICS: What are CESM xml variables and CESM environment variables?

Like in CESM1.0 and CESM1.1, CESM1.2 cases are customized, built and run largely through setting what CESM calls "environment variables". These actually appear to the user as variables defined in xml files. Those files appear in the case directory once a case is created and are named something like env_*.xml. They are converted to actual environment variables via a csh script called ccsm_getenv. That script calls a perl script called xml2env that converts the xml files to shell files that are then sourced and removed. The ccsm_getenv and xml2env exist in the $CASEROOT/Tools directory. The environment variables are specified in xml files to support extra automated error checking and automatic generation of env variable documentation. If you want to have the cesm environment variables in your local shell environment, do the following

```
> cd $CASEROOT
> source ./Tools/ccsm_getenv
```

You must run the ccsm_getenv from the CASEROOT directory exactly as shown above. There are multiple env_*.xml files including env_case.xml, env_mach_pes.xml, env_build.xml, and env_run.xml. To a large degree, the different env files exist so variables can be locked in different phases of the case setup, build, and run process. For more info on locking files, see the Section called *BASICS: Why is there file locking and how does it work?*. The important point is that env_case.xml variables cannot be changed after create_newcase is invoked. env_mach_pes cannot be changed after **cesm_setup** is invoked unless you plan to invoke the commands **cesm_setup -clean**, and **cesm_setup** again. env_build

variables cannot be changed after the model is built unless you plan to clean and rebuild. env_run.xml[2] variables can be changed at any time. The CESM scripting software checks that xml files are not changed when they shouldn't be.

CESM recommends using the xmlchange tool to modify env variables. This will decrease the chance that typographical errors will creep into the xml files. Conversion of the xml files to environment variables can fail silently with certain xml format errors. To use xmlchange, do, for instance,

```
> cd $CASEROOT
> ./xmlchange STOP_OPTION=nmonths
> ./xmlchange STOP_N=6
```

which will change the variables STOP_OPTION and STOP_N in the file env_run.xml to the specified values. The xml files can be edited manually, but users should take care not to introduce any formatting errors that could lead to incomplete env settings. If there appear to be problems with the env variables (i.e. if the model doesn't seem to have consistent values compared to what's set in the xml files), then confirm that the env variables are being set properly. There are a couple of ways to do that. First, run the ccsm_getenv script as indicated above and review the output generated by the command "env | sort". The env variables should match the xml settings. Another option is to edit the $CASEROOT/Tools/ccsm_getenv script and comment out the line "rm $i:r". That should leave the shell env files around, and they can then be reviewed. The latter approach should be undone as soon as possible to avoid problems running ccsm_getenv later.

BASICS: How do I modify the value of CESM xml variables?

CESM recommends using the xmlchange tool to modify env variables. xmlchange supports error checking as part of the implementation. Also, using xmlchange will decrease the chance that typographical errors will creep into the xml files. Conversion of the xml files to environment variables can fail silently with certain xml format errors. To use xmlchange, do, for instance,

```
> cd $CASEROOT
> ./xmlchange STOP_OPTION=nmonths
> ./xmlchange STOP_N=6
```

which will change the variables STOP_OPTION and STOP_N in the file env_run.xml to the specified values. The xml files can be edited manually, but users should take care not to introduce any formatting errors that could lead to incomplete env settings. See also .

BASICS: Why aren't my $CASEROOT xml variable changes working?

It's possible that a formatting error has been introduced in the env xml files. This would lead to problems in setting the env variables. If there appear to be problems with the env variables (i.e. if the model doesn't seem to have consistent values compared to what's set in the xml files), then confirm that the env variables are being set properly. There are a couple of ways to do that. First, run the ccsm_getenv script via

```
> cd $CASEROOT
> source ./Tools/ccsm_getenv
> env
```

and review the output generated by the command "env". The env variables should match the xml settings. Another option is to edit the $CASEROOT/Tools/ccsm_getenv script and comment out the line "rm $i:r". That

should leave the shell env files around, and they can then be reviewed. The latter approach should be undone as soon as possible to avoid problems running ccsm_getenv later.

BASICS: How do I run multiple cases all using a single executable?

In CESM, the directory containing the model executable is cleanly separated from the directory where the model is run. As a result, it is now straightforward to run multiple cases *where the env_build.xml for each case is identical* all using a pre-built executable. As an example, this type of flexibility greatly simplifies carrying out UQ analysis.

The following outlines the steps involved to do this.

1. Create the executable that all runs will use. Call this case RefExe. The following would be a sample **create_newcase** command:

```
> cd $CCSMROOT/scripts
> create_newcase -case RefExe -compset B1850CN -res ne30_g16 -mach hopper
> cd RefExe
> ./cesm_setup
> ./RefExe.build
```

   Verify that the model has build successfully. For reference below - the $EXEROOT for the RefExe case will be `/scratch/scratchdirs/$CCSMUSER/RefExe/bld`.

2. All subsequent calls to create_newcase that will use the RefExe executable must have identical arguments for -compset, -res and -mach. Lets say that you want to run 2 separate cases, RefExe_Case1 and RefExe_Case2 that both use the executable RefExe. You would then do the following:

```
> cd $CCSMROOT/scripts
> create_newcase -case RefExe_Case1 -compset B1850CN -res ne30_g16 -mach hopper
> cd RefExe_Case1
> ./cesm_setup
> xmlchange EXEROOT=/scratch/scratchdirs/$CCSMUSER/RefExe/bld.
> xmlchange BUILD_COMPLETE=TRUE
> qsub RefExe_Case1.run
```

```
> cd $CCSMROOT/scripts
> create_newcase -case RefExe_Case2 -compset B1850CN -res ne30_g16 -mach hopper
> cd RefExe_Case1
> ./cesm_setup
> xmlchange EXEROOT=/scratch/scratchdirs/$CCSMUSER/RefExe/bld.
> xmlchange BUILD_COMPLETE=TRUE
> qsub RefExe_Case2.run
```

Note that by setting BUILD_COMPLETE in `env_build.xml` to TRUE, the scripts assume that the model has already been built for the case. Normally, the $CASE.build script fills this in when the build is successful. However, since you will not invoke the build for RefExe_Case1 and RefExe_Case2, you must then manually tell the script where the build is - and that it has been successful. This option is FOR EXPERTS ONLY and should only be used by those users that are completely familiar with the CESM scripts.

BASICS: How do I use the ESMF library and ESMF interfaces?

CESM supports use of either the CESM designed component interfaces which are based on MCT datatypes and are used by default in CESM or ESMF compliant component interfaces. In both cases, the driver and component models remain fundamentally the same. The ESMF interface implementation exists in CESM to support further development and testing of an ESMF driver or ESMF couplers and to allow CESM model components to interact with other coupled systems using ESMF coupling standards.

ESMF is NOT required or provided by CESM. It must be downloaded[3] and installed separately. It is safest to compile ESMF and CESM with identical compilers and mpi versions. It may be possible to use versions that are different but compatible; however, it is hard to predict which versions will be compatible and using different versions can result in problems that are difficult to track down.

There are three possible modes of interaction between CESM and ESMF.

1. No linking to an external ESMF library. CESM uses a native implementation of ESMF timekeeping interfaces (default).

   To run with the MCT interfaces and the native time manager, set the following env variables

   ```
   - cd to your case directory
   - edit env_build.xml
     - set COMP_INTERFACE to "MCT"
     - set USE_ESMF_LIB to "FALSE"
   ```

2. Linking with an ESMF library to use the ESMF time manager but continued use of the native CESM component interfaces.

   To run with the native interfaces and ESMF time manager, set the following env variables

   ```
   - cd to your case directory
   - edit env_build.xml
     - set COMP_INTERFACE to "MCT"
     - set USE_ESMF_LIB to "TRUE"
     - set ESMF_LIBDIR to a valid installation directory of ESMF version 5.3.0
   ```

3. Linking with an ESMF library in order to use ESMF component interfaces. In this mode ESMF timekeeping is also activated.

   To run with the ESMF interfaces and ESMF time manager, set the following env variables

   ```
   - cd to your case directory
   - edit env_build.xml
     - set COMP_INTERFACE to "ESMF"
     - set USE_ESMF_LIB to "TRUE"
     - set ESMF_LIBDIR to a valid installation directory of ESMF version 5.3.0
   ```

The ESMF library can be activated in two ways in CESM. The primary way is via the ESMF_LIBDIR env variable in the env_build.xml file described above. The secondary way is via a system environment variable called ESMFMKFILE. If this environment variable is set either through a system or module command, then the ESMF library will be picked up by the CESM scripts, but the local CESM variable, ESMF_LIBDIR, will always have precedence.

To verify the correctness of the ESMF component interfaces in CESM, compute and compare CESM global integrals with identical runs differing only in the use of the MCT and ESMF interfaces. In both cases, the ESMF library should be active to guarantee identical time manager values. In both runs, the 'INFO_DBUG' parameter in env_run.xml should be set to 2 which activates the global integral diagnostics. A valid comparison would be a 10 day test from the same initial conditions. The global

integrals produced in the cpl log file should be identical in both cases. This test can be set up manually as described above or a CME test can be carried out which is designed to test this exact capability.

BASICS: Why is there file locking and how does it work?

In CESM, there are several different $CASEROOT xml files. These include env_case.xml, env_mach_pes.xml, env_build.xml, and env_run.xml. These files are organized so that variables can be locked during different phases of the case setup, build, and run. Locking variables is a feature of CESM that prevents users from changing variables after they have been resolved (used) in other parts of the scripts system. The variables in env_case are locked when create_newcase is called. The env_mach_pes variables are locked when **cesm_setup** is called. The env_build variables are locked when CESM is built, and the env_run variables are never locked and can be changed anytime. In addition, the Macros file is locked as part of the build step. The $CASEROOT/LockedFiles directory saves copies of the xml files to facilitate the locking feature. In summary:

- `env_case.xml` is locked upon invoking **create_newcase** and cannot be unlocked. To change settings in env_case, a new case has to be generated with create_newcase.
- `env_mach_pes.xml` is locked after running **cesm_setup**. After changing variable values in this file, you need to invoke **cesm_setup -clean** and then **cesm_setup**.
- `Macros` and `env_build.xml` are locked upon the *successful* completion of **$CASE.build**. Both `Macros` and `env_build.xml` can be unlocked by invoking **$CASE.cleanbuild** and then the model should be rebuilt.

BASICS: What are the directories and files in my case directory?

The following describes many of the files and directories in the $CASEROOT directory.

Buildconf/

is the directory where the buildnml and buildexe component scripts reside and where the input_data_list files are generated by the buildnml scripts.

CaseDocs/

is the directory where copies of the latest namelist/text input files from invoking **preview_namelists** are placed. These files should not be edited and exist only to help document the case setup and run.

SourceMods/

contains directories for each component where case specific source code modifications can be included. The source files in these directories will always be used in preference to the source code in $CCSMROOT. This feature allows users to modify CESM source code on a case by case basis if that is preferable to making modifications in the $CCSMROOT sandbox.

LockedFiles/

is the directory that holds copies of the locked files.

Macros

is the Makefile Macros file for the current configuration. The Makefile is located in the Tools directory and is identical on all machines. The Macros file is a machine and compiler dependent file. This file is locked during the build step.

README.case

provides a summary of the commands used to generate this case.

$CASE.build

is the script that is run interactively to build the CESM model.

$CASE.clean_build

is the script that cleans the CESM build.

$CASE.l_archive

is the script that is submitted to the batch queue to archive CESM data to the long-term archive storage system, like an hpss or mass storage system.

$CASE.run

is the script that is submitted to the batch queue to run a CESM job. This script could also be run interactively if resources allow.

$CASE.submit

is the script that will submit the job to the system's particular batch queuing system.

check_input_data

is a tool that checks for missing input datasets and provides a capability for exporting them to local disk.

cesm_setup

is the script that is run to generate the $CASE.run script for the target env_mach_pes.xml file and if they have not already been created, the user_nl_xxx files for the target components.

create_production_test

is a tool that generates an exact restart test in a separate directory based on the current case.

env_*.xml files

contain variables used to set up, build, and run CESM.

logs/

is the directory that contains a copy of the component log files from successful case runs.

timing/

is the directory that contains timing output from each successful case run.

xmlchange

is a utility that supports changing xml variables in the $CASEROOT xml files.

$CASEROOT/Tools/

a directory containing many scripts that are used to set up the CESM model as well as run it. Some of particular note are

- Makefile is the Makefile that will be used for the build.
- cesm_buildexe is invoked by $CASEROOT/$CASE.build to generate the model executable. This script calls the component buildexe scripts in Buildconf.

- cesm_buildnml is invoked by $CASEROOT/$CASE.build to generate the component namelists in $RUNDIR. This script calls the component buildnml scripts in Buildconf.
- ccsm_check_lockedfiles checks that any files in the $CASEROOT/LockedFiles/ directory match those in the $CASEROOT directory. This helps protect users from overwriting variables that should not be changed.
- ccsm_getenv converts the xml variables in $CASEROOT to csh environmental variables.
- getTiming.csh generates the timing information.
- getTiming2.pl generates timing information and is used by getTiming.csh.
- mkDepends generates Makefile dependencies in a form suitable for inclusion into a Makefile.
- st_archive.sh is the short-term archive script. It moves model output out of run directory to the short-term archive directory. Associated with DOUT_S and DOUT_S_ROOT env variables in env_run.xml.
- taskmaker.pl derives pe counts and task and thread geometry info based on env var values set in the env_mach_pes file.
- xml2env converts env_*xml files to shell environment variable files that are then sourced for inclusion in the model environment. Used by the ccsm_getenv script.

IO: What is pio?

The parallel IO (PIO) library is included with CESM and is automatically built as part of the CESM build. CESM components use the PIO library to read and/or write data. The PIO library is a set of interfaces that support serial netcdf, parallel netcdf, or binary IO transparently. The implementation allows users to easily modify the pio setup on the fly to change the method (serial netcdf, parallel netcdf, or binary data) as well as various parameters associated with PIO to optimize IO performance.

CESM prefers that data be written in CF compliant netcdf format to a single file that is independent of all parallel decomposition information. Historically, data was written by gathering global arrays on a root processor and then writing the data from the root processor to an external file using serial netcdf. The reverse process (read and scatter) was done for reading data. This method is relatively robust but is not memory scalable, performance scalable, or performance flexible.

PIO works as follows. The PIO library is initialized and information is provided about the method (serial netcdf, parallel netcdf, or binary data), and the number of desired IO processors and their layout. The IO parameters define the set of processors that are involved in the IO. This can be as few as one and as many as all available processors. The data, data name and data decomposition are also provided to PIO. Data is written through the PIO interface in the model specific decomposition. Inside PIO, the data is rearranged into a block decomposition on the IO processors and the data is then written serially using netcdf or in parallel using pnetcdf. There are several namelist options to control PIO functionality. Refer to the Parallel I/O (PIO) control variables[4] in the env_run namelist documentation for details.

There are several benefits associated with using PIO. First, even with serial netcdf, the memory use can be significantly decreased because the global arrays are decomposed across the IO processors and written in chunks serially. This is critical as CESM runs at higher resolutions where global arrays need to be minimized due to memory availability. Second, pnetcdf can be turned on transparently potentially improving the IO performance. Third, PIO parameters such as the number of IO tasks and their layout can be tuned to reduce memory and optimize performance on a machine by machine basis. Fourth, the standard global gather and write or read and global scatter can be recovered by setting the number of io tasks to 1 and using serial netcdf.

CESM uses the serial netcdf implementation of PIO and pnetcdf is turned off in PIO by default. Several components provide namelist inputs that allow use of pnetcdf in PIO. To use pnetcdf, a pnetcdf library (like netcdf) must be available on the local

machine and PIO pnetcdf support must be turned on when PIO is built. This is done as follows

1. Locate the local copy of pnetcdf. We recommend version 1.3.1 (1.2.0 or newer is required)
2. Set PNETCDF_PATH in the Macros file to the directory of the pnetcdf install (ie. /contrib/pnetcdf1.3.1/).
3. Run the clean_build script if the model has already been built.
4. Run the build script to rebuilt pio and the full CESM system.
5. Change component IO namelist settings to pnetcdf and set appropriate IO tasks and layout.

There is an ongoing effort between CESM, pio developers, pnetcdf developers and hardware vendors to understand and improve the IO performance in the various library layers. To learn more about pio, see the pio documentation.[5]

IO: How do I use pnetcdf?

See the Section called *IO: What is pio?*

CAM: How do I customize CAM output fields?

In this example, we further modify our EXAMPLE_CASEp code to set various CAM output fields. The variables that we set are listed below. See CAM Namelist Variables[6] for a complete list of CAM namelist variables.

`avgflag_pertape`

Sets the averaging flag for all variables on a particular history file series. Default is to use default averaging flags for each variable. Average (A), Instantaneous (I), Maximum (X), and Minimum (M).

`nhtfrq`

Array of write frequencies for each history files series.

When NHTFRQ(1) = 0, the file will be a monthly average. Only the first file series may be a monthly average.

When NHTFRQ(i) > 0, frequency is input as number of timesteps.

When NHTFRQ(i) < 0, frequency is input as number of hours.

`mfilt`

Array of number of time samples to write to each history file series (a time sample is the history output from a given timestep).

`ndens`

Array specifying output format for each history file series. Valid values are 1 or 2. '1' implies output real values are 8-byte and '2' implies output real values are 4-byte. Default: 2,2,2,2,2,2

`fincl1 = 'field1', 'field2', ...`

List of fields to add to the primary history file.

`fincl[2..6] = 'field1', 'field2', ...`

List of fields to add to the auxiliary history file.

`fexcl1 = 'field1', 'field2', ...`

List of field names to exclude from the default primary history file (default fields on the Master Field List).

`fexcl[2..6] = 'field1', 'field2',...`

List of the field names to exclude from the auxiliary history files.

Edit `user_nl_cam` and add the following lines at the end of the file:

```
avgflag_pertape = 'A','I'
nhtfrq = 0 ,-6
mfilt  = 1 , 30
ndens  = 2 , 2
fincl1 = 'FSN200','FSN200C','FLN200',
         'FLN200C','QFLX','PRECTMX:X','TREFMXAV:X','TREFMNAV:M',
         'TSMN:M','TSMX:X'
fincl2 = 'T','Z3','U','V','PSL','PS','TS','PHIS'
```

avgflag_pertape specifies how the output data will be averaged. In the first output file, `b40.2000p.cam2.h0.yyyy-mm.nc`, data will be averaged monthly. In the second output file, `b40.2000p.cam2.h1.yyyy-mm-dd.nc`, data will be instantaneous.

nhtfrq sets the frequency of data writes, so `b40.2000p.cam2.h0.yyyy-mm.nc` will be written as a monthly average, while `b40.2000p.cam2.h1.yyyy-mm-dd.nc` will contain time slices that are written every 6 hours.

mfilt sets the model to write one time sample in `b40.2000p.cam2.h0.yyyy-mm.nc` and 30 time samples in `b40.2000p.cam2.h1.yyyy-mm-dd.nc`.

ndens sets both files to have 32-bit netCDF format output files.

fincl1 sets the output fields for b40.2000p.cam2.h0.yyyy-mm.nc. A complete list of the CAM output fields appears here. In this example, we've asked for more variables than will fit on a Fortran line. As you can see, it is all right to split variable lists across lines. Also in this example, we've asked for maximum values of TREFMXAV and TSM, and minimum values of TREFMNAV and TSMN.

fincl2 sets the output fields for b40.2000p.cam2.h1.yyyy-mm-dd.nc, much the same as fincl1 sets output fields for b40.2000p.cam2.h0.yyyy-mm.nc, only in this case, we are asking for instantaneous values rather than averaged values, and choosing different output fields.

CAM: How do I customize CAM forcings?

To set the greenhouse gas forcings, you must first understand the namelist variables associated with them. See CAM Namelist Variables[7] for a complete list of CAM namelist variables.

`scenario_ghg`

Controls treatment of prescribed co2, ch4, n2o, cfc11, cfc12 volume mixing ratios. May be set to 'FIXED' or 'RAMPED' or 'RAMP_CO2_ONLY'.

FIXED => volume mixing ratios are fixed and have either default or namelist input values.

RAMPED => volume mixing ratios are time interpolated from the dataset specified by bndtvghg.

RAMP_CO2_ONLY => only co2 mixing ratios are ramped at a rate determined by the variables ramp_co2_annual_rate, ramp_co2_cap, and ramp_co2_start_ymd.

Default: FIXED

`bndtvghg`

Full pathname of time-variant boundary dataset for greenhouse gas surface values.

`rampyear_ghg`

If scenario_ghg is set to "RAMPED" then the greenhouse gas surface values are interpolated between the annual average values read from the file specified by bndtvghg. In that case, the value of this variable (> 0) fixes the year of the lower bounding value (i.e., the value for calendar day 1.0) used in the interpolation. For example, if rampyear_ghg = 1950, then the GHG surface values will be the result of interpolating between the values for 1950 and 1951 from the dataset. Default: 0

Edit `user_nl_cam` and add the following lines at the end of the file. The following assumes that "my_inputdata_path" is identical to $DIN_LOC_ROOT.

```
scenario_ghg = 'RAMPED'
bndtvghg = 'my_inputdata_path/atm/cam/ggas/ghg_hist_1765-2005_c091218.nc'
rampyear_ghg = 2000
```

CAM/CLM: How do I change history file output frequency and content for CAM and CLM during a run?

If you want to change the frequency of output for CAM or CLM (i.e. generate output every 6 model hours instead of once a model day) in the middle of a run, or if you want to change the fields that are output, in the middle of a run, you need to stop the run, rebuild and rerun it with the same casename and branch from the same casename. See the steps below for doing a branch run while retaining the casename.

Rebuilding the case and restarting it where you left off, are necessary because CAM and CLM only read namelist variables once, at the beginning of a run. This is not the case for POP and CICE, they read the namelist input on every restart, and therefore for POP and CICE, you can change output fields and frequency by modifying the appropriate namelist variables and then doing a restart.

The following example shows case B40.20th.1deg which runs from 1850 to 2005, and will generate high frequency output for years 1950 through 2005. CAM will output data every six hours instead of once a day. Also starting at year 1950 additional fields will be output by the model.

1. The first step is to create case b40.20th.1deg and run the case for years 1850 through 1949 with your initial settings for output.
2. Next move your entire case directory, $CASEDIR, somewhere else, because you need to rebuild and rerun the case using the same name.

   ```
   > cd $CASEDIR
   > mv b40.20th.1deg b40.20th.1deg.1850-1949
   ```

3. Now move your run directory, $RUNDIR, somewhere else as well.

   ```
   > cd $RUNDIR
   > mv b40.20th.1deg b40.20th.1deg.1850-1949
   ```

4. Next create a new case in your case directory with the same name, b40.20th.1deg.

```
> cd $CASEDIR/scripts
> create_newcase -mach yellowstone -compset B_1850-2000_CN -res f09_g16 -case b40.
cd $RUNDIR
```

5. Next invoke the following commands

```
> cd $CASEROOT
> xmlchange RUN_TYPE='branch'
> xmlchange RUN_REFCASE='b40.20th.1deg'
> xmlchange RUN_REFDATE='1948-01-01'
> xmlchange CAM_NML_USE_CASE='1850-2005_cam4'
> xmlchange BRNCH_RETAIN_CASENAME='TRUE'
> xmlchange GET_REFCASE='FALSE'
```

6. Next set up the case and edit the coupler and CAM and CLM namelists.

   a. Set up the case.

   ```
   > ./cesm_setup
   ```

   b. Edit user_nl_cpl. Add the following to the end of the file. brnch_retain_casename = .true.

   c. Edit user_nl_cam. Check that bndtvghg = '$DIN_LOC_ROOT' and add the following to the end of the file

   ```
   doisccp = .true.
   isccpdata = '/fis/cgd/cseg/csm/inputdata/atm/cam/rad/isccp.tautab_invt
   mfilt   = 1,365,30,120,240
   nhtfrq  = 0,-24,-24,-6,-3
   fincl2  = 'TREFHTMN','TREFHTMX','TREFHT','PRECC','PRECL','PSL'
   fincl3  = 'CLDICE','CLDLIQ','CLDTOT','CLOUD','CMFMC','CMFMCDZM','FISC
             'FLDS','FLDSC','FLNS','FLUT','FLUTC','FSDS','FSDSC','FSNS',
             'FSNSC','FSNTOA','FSNTOAC','LHFLX','OMEGA','OMEGA500',
             'PRECSC','PRECSL','PS','Q','QREFHT','RELHUM','RHREFHT','SHFl
             'SOLIN','T','TGCLDIWP','TGCLDLWP','U','V','Z3'
   fincl4  = 'PS:I','PSL:I','Q:I','T:I','U:I','V:I','Z3:I'
   fincl5  = 'CLDTOT','FLDS','FLDSC','FLNS','FLNSC','FSDS','FSDSC','FSNS
             'LHFLX','PRECC','PRECL','PRECSC','PRECSL','SHFLX',
             'PS:I','QREFHT:I','TREFHT:I','TS:I'
              /
   ```

   d. Edit user_nl_clm. This adds four auxilary history files in addition to the standard monthly files. The first two are daily, and the last two are six and three hourly.

   ```
   hist_mfilt   = 1,365,30,120,240
   hist_nhtfrq  = 0,-24,-24,-6,-3
   hist_fincl2  = 'TSOI', 'TG',   'TV',   'FIRE',   'FSR', 'FSH', 'EFLX_LH
   hist_fincl3  = 'FSA'
   hist_fincl4  = 'TSOI', 'TG',   'TV',   'FIRE',   'FSR', 'FSH', 'EFLX_LH
   hist_fincl5  = 'TSOI', 'TG',   'TV',   'FIRE',   'FSR', 'FSH', 'EFLX_LH
   ```

7. Now build and run the case.

```
> b40.20th.1deg.build
> bsub < b40.20th.1deg.run
```

CAM: How do I use B compset history output to create SST/ICE data files to drive an F compset?

The following was contributed by Art Mirin and outlines the procedure you would use if you want to run an F-configuration case forced by monthly averages of SST and ice coverage from a B-configuration case. As an example, the following uses an f09_g16 CESM B-configuration simulation using CAM5 physics and with cosp enabled. The procedure to create the SST/ICE file is as follows:

1. Save monthly averaged 'aice' information from cice code (this is the default).
2. Save monthly averaged SST information from pop2. To do this, copy $CCSMROOT/models/ocn/pop2/input_templates/gx1v6_tavg_contents, to $CASEROOT/SourceMods/src.pop2 and change the 2 in front of SST to 1 for monthly frequency.
3. Extract (using ncrcat) SST from monthly pop2 history files and form a single netcdf file containing just SST; change SST to SST_cpl.

   ```
   > ncrcat -v SST case.pop.h.*.nc temp.nc
   > ncrename -v SST,SST_cpl temp.nc sst_cpl.nc
   ```

4. Extract aice from monthly cice history files and form a single netcdf file containing aice; change aice to ice_cov; divide values by 100 (to convert from percent to fraction).

   ```
   > ncrcat -v aice case.cice.h.*.nc temp.nc
   > ncrename -v aice,ice_cov temp.nc temp2.nc
   > ncap2 -s 'ice_cov=ice_cov/100.' temp2.nc ice_cov.nc
   ```

5. Modify fill values in the sst_cpl file (which are over land points) to have value -1.8 and remove fill and missing value designators; change coordinate lengths and names: to accomplish this, first run ncdump, then replace _ with *-1.8* in SST_cpl, then remove lines with _FillValue and missing_value. (Note: although it might be possible to merely change the fill value to -1.8, this is conforming to other SST/ICE files, which have SST_cpl explicitly set to -1.8 over land.) To change coordinate lengths and names, replace nlon by lon, nlat by lat, TLONG by lon, TLAT by lat. The last step is to run ncgen. Note: when using ncdump followed by ncgen, precision will be lost; however, one can specify -d 9,17 to maximize precision - as in the following example:

   ```
   > ncdump -d 9,17 old.nc > old
   > ncgen -o new.nc new
   ```

6. Modify fill values in the ice_cov file (which are over land points) to have value 1 and remove fill and missing value designators; change coordinate lengths and names; patch longitude and latitude to replace missing values: to accomplish this, first run ncdump, then replace _ with *1* in ice_cov, then remove lines with _FillValue and missing_value. To change coordinate lengths and names, replace ni by lon, nj by lat, TLON by lon, TLAT by lat. To patch longitude and latitude arrays, replace values of those arrays with those in sst_cpl file. The last step is to run ncgen. (Note: the replacement of longitude and latitude missing values by actual values should not be necessary but is safer.)
7. Combine (using ncks) the two netcdf files.

   ```
   > ncks -v ice_cov ice_cov.nc sst_cpl.nc
   ```

   Rename the file to ssticetemp.nc. The time variable will refer to the number of days at the end of each month, counting from year 0, whereas the actual simulation began at year 1 (CESM default); however, we want time values to be in the middle of each month, referenced to the first year of the simulation (first time value equals 15.5); extract (using ncks) time variable from existing amip sst file (for correct number of months - 132 in this example) into working netcdf file.

   ```
   > ncks -d time,0,131 -v time amipsst.nc ssticetemp.nc
   ```

Add date variable: ncdump date variable from existing amip sst file; modify first year to be year 0 instead of 1949 (do not including leading zeroes or it will interpret as octal) and use correct number of months; ncgen to new netcdf file; extract date (using ncks) and place in working netcdf file.

```
> ncks -v date datefile.nc ssticetemp.nc
```

Add datesec variable: extract (using ncks) datesec (correct number of months) from existing amip sst file and place in working netcdf file.

```
> ncks -d time,0,131 -v datesec amipsst.nc ssticetemp.nc
```

8. At this point, you have an SST/ICE file in the correct format. However, due to CAM's linear interpolation between mid-month values, you need to apply a procedure to assure that the computed monthly means are consistent with the input data. To do this, you can invoke the bcgen code in models/atm/cam/tools/icesst and following the following steps:
   a. Rename SST_cpl to SST, and ice_cov to ICEFRAC in the current SST/ICE file:

      ```
      > ncrename -v SST_cpl,SST -v ice_cov,ICEFRAC ssticetemp.nc
      ```

   b. In driver.f90, sufficiently expand the lengths of variables prev_history and history (16384 should be sufficient); also comment out the test that the climate year be between 1982 and 2001 (lines 152-158).
   c. In bcgen.f90 and setup_outfile.f90, change the dimensions of xlon and ???TODO xlat to (nlon,nlat); this is to accommodate use of non-cartesian ocean grid.
   d. In setup_outfile.f90, modify the 4th and 5th ???TODO arguments in the calls to wrap_nf_def_var for *lon* and *lat* to be *2* and *dimids*; this is to accommodate use of non-cartesian ocean grid.
   e. Adjust Makefile to have proper path for LIB_NETCDF and INC_NETCDF.
   f. Modify namelist accordingly.
   g. Make bcgen and execute per instructions. The resulting sstice_ts.nc file is the desired ICE/SST file.

9. Place new SST/ICE file in desired location. In the $CASEROOT for the F compset you create, modify env_run.xml to have :
   a. SSTICE_DATA_FILENAME point to the complete path of your SST/ICE file.
   b. SSTICE_GRID_FILENAME correspond to full path of (in this case) gx1v6 grid file.
   c. SSTICE_YEAR_START set to 0, and SSTICE_YEAR_END to one less than the total number of years; set SSTICE_YEAR_ALIGN to 1 (since CESM starts counting at year 1).

POP/CICE: How are CICE and POP decompositions set and how do I override them?

The pop and cice models both have similar decompositions and strategies for specifying the decomposition. Both models support decomposition of the horizontal grid into two-dimensional blocks, and these blocks are then allocated to individual processors inside each component. The decomposition must be specified when the models are built. There are four environment variables in env_build.xml for each model that specify the decomposition used. These variables are POP or CICE followed by _BLCKX, _BLCKY, _MXBLCKS, and _DECOMP. BLCKX and BLCKY specify the size of the local block in grid cells in the "x" and "y" direction. MXBLCKS specifies the

maximum number of blocks that might be on any given processor, and DECOMP specifies the strategy for laying out the blocks on processors.

The values for these environment variables are set automatically by the scripts in the $CASEROOT/Buildconf directory whenever the model is built or run is run. The scripts that generate the decompositions are CASEROOT/Buildconf/generate_pop_decomp.pl and $CASEROOT/Buildconf/generate_cice_decomp.pl. Those tools leverage decompositions stored in xml files, $CCSMROOT/models/ocn/pop2/bld/pop_decomp.xml and $CCSMROOT/models/ice/cice/bld/cice_decomp.xml, respectively. These utilities set the decomposition for a given resolution and total processor count. The decomposition used can have a significant effect on the model performance, and the decompositions specified by the tools above generally provide optimum or near optimum values for the given resolution and processor count. More information about cice and pop decompositions can be found in each of those user guides.

The decompositions can be specified manually by setting the environment variable POP_AUTO_DECOMP or CICE_AUTO_DECOMP to false in env_build.xml (which turns off use of the scripts above) and then setting the four BLCKX, BLCKY, MXBLCKS, and DECOMP environment variables in env_build.xml.

In general, relatively square and evenly divided Cartesian decompositions work well for pop at low to moderate resolution. Cice performs best with "tall and narrow" blocks because of the load imbalance for most global grids between the low and high latitudes. At high resolutions, more than one block per processor can result in land block elimination and non-Cartesian decompositions sometimes perform better. Testing of several decompositions is always recommended for performance and validation before a long run is started.

POP: How do I initialize POP2 with a spun-up initial condition?

The startup/spunup initialization option is a specialized active-ocean model suboption available in the CESM1.1 POP2 model which can be used only in conjunction with a CESM "startup" case; it is not designed to work with "hybrid" or "branch" cases.

The recommended method for initializing the CESM active ocean model (POP2) in a CESM startup case is to use the default settings; these initialize the ocean model from Levitus initial conditions and a state of rest. Occasionally, however, researchers are interested in a startup run in which only the ocean model is initialized from a "spun up" ocean condition generated from a previous CESM run. To accommodate their request, a nonstandard method of initializing POP2 in a startup case was developed. It is called the startup_spunup option. It is a research option that is designed for use by expert users only.

Because of the complex interactions between the ocean-model parameterizations used to generate the spun-up case and those used in the new startup case, it is impossible to provide a single recommended spun-up ocean initial condition for all circumstances. Instead, researchers must carefully select an existing solution whose case conditions closely match those in the new case. A mismatch of options between the spun-up case and the new case can result in scientifically invalid solutions.

When a startup_spunup case is necessary, use this procedure:

1. Currently, the default RUN_TYPE XML variable is set to "hybrid". User's will need to change the RUN_TYPE to "startup" after running create_newcase using the xmlchange command as follows:

```
> create_newcase -case ~/cesm/EXAMPLE_CASEocn \
                 -mach yellowstone \
                 -compset B20TR \
                 -res 0.9x1.25_gx1v6
```

```
> cd ~/cesm/EXAMPLE_CASEocn
> xmlchange -file env_run.xml -id RUN_TYPE -val startup
> ./cesm_setup
```

2. The ocean restart filename is of the form ${CASE_SP}.pop.r.$date, where $date is the model date of your spun-up dataset. If the ocean restart files were written in binary format, a companion ascii-formatted restart "header" file will also exist. The companion header file will have the same name as the restart file, except that it will have the suffix ".hdr" appended at the end of the filename. You must copy both the binary restart file and the header file to your data directory.
3. The spun-up ocean restart and restart header files must be available to your new case. Copy them directly into $RUNDIR. It is critically important to copy both the binary restart file and its companion header file to the $RUNDIR.

   ```
   > cp ${CASE_SP}.pop.r.$date       $RUNDIR
   > cp ${CASE_SP}.pop.r.${date}.hdr $RUNDIR
   ```

4. Redefine the ocean-model initial-condition dataset by editing `user_nl_pop2` and add the following lines at the end of the file (enter the resolved string for ${CASE_SP}).

   ```
   set init_ts_suboption = 'spunup'
   init_ts_file = '${CASE_SP}.pop.r.$date
   ```

   Note that the model will automatically look for the ${CASE_SP}.pop.r.${date}.hdr file in $RUNDIR.
5. Build and run as usual.

DRIVER: Is there more information about the coupler/driver implementation?

Additional implementation details are provided in the the CESM coupler user guide[8] about sequencing, parallel IO, performance, grids, threading, budgets, and other items.

DRIVER: How do I pass in new fields between components?

In CESM, coupler code has been improved in order to remove the need to change any coupler code when adding the exchange of new fields between model components. To accomplish this, a new standardized naming convention has been introduced for field names that are exchanged between model components. This is summarized below.

```
================================================================
   New standardized naming convention
================================================================

  ---------
  definitions:
  ---------
  state-prefix
    first 3 characters: Sx_, Sa_, Si_, Sl_, So_
    one letter indices: x,a,l,i,o,s,g,r
    x => coupler (mapping, merging, atm/ocn flux calc done on coupler procs)
    a => atm
    l => lnd
    i => ice
    o => ocn
    g => glc
    s => snow (from clm to glc)
    r => rof
```

```
state-name
  what follows state prefix

flux-prefix
  first 5 characters: Flmn__
  lm => between components l and m
  n  => computed by component n
  example: Fioi => ice/ocn flux computed by ice
  example: Fall => atm/lnd flux computed by lnd
  If flux prefix has first letter of P (so first five characters are PFlmn_)
  then flux is passed straight through without scaling by the corresponding fraction)

flux-name
  what follows flux-prefix

---------
rules:
---------
1) states:
   a) atm attributes fields that HAVE a state-prefix of Sx_ in seq_flds_x2a_states
      rule: will merge all identical values of the state-names from
         seq_flds_i2x_states
         seq_flds_l2x_states
         seq_flds_o2x_states
         seq_flds_xao_states
       to obtain output state-name in seq_flds_x2a_states

      rule: to merge input states that originate in the
         lnd (l2x_a) will be scaled by the lndfrac
         ice (i2x_a) will be scaled by the icefrac
         cpl (xao_a) will be scaled by the ocnfrac
         ocn (o2x_a) will be scaled by the ocnfrac

      example:
         seq_flds_l2x_states = "Sl_t"
         seq_flds_i2x_states = "Si_t"
         seq_flds_o2x_states = "So_t"
         seq_flds_x2a_states = "Sx_t"
         attribute fields Sl_t, Si_t, So_t, in
         attribute vectors l2x_a, i2x_a, o2x_a will be
         merged to obtain attribute Sx_t in attribute vector x2a_a

   b) atm attribute fields that DO NOT HAVE a state-prefix of Sx_ in seq_flds_x2a_sta
      rule: copy directly all variables that identical state-prefix
            AND state-name in
         seq_flds_i2x_states and seq_flds_x2a_states
         seq_flds_l2x_states and seq_flds_x2a_states
         seq_flds_o2x_states and seq_flds_x2a_states
         seq_flds_xao_states and seq_flds_x2a_states

      example
         seq_flds_i2x_states = ":Si_snowh"
         seq_flds_x2a_states = ":Si_snowh"
         attribute field of Si_snowh in i2x_a will be copied to
         attribute field Si_snowh in x2a_a

2) fluxes:
   rule: will merge all identical values of the flux-names from
       seq_flds_i2x_states
       seq_flds_l2x_states
       seq_flds_o2x_states
       seq_flds_xao_states
     to obtain output state-name in seq_flds_x2a_states

   rule: input flux fields that originate in the
```

```
         lnd (l2x_a) will be scaled by the lndfrac
         ice (i2x_a) will be scaled by the icefrac
            - ignore all fluxes that are ice/ocn fluxes (e.g. Fioi_)
         cpl (xao_a) will be scaled by the ocnfrac
         ocn (o2x_a) will be scaled by the ocnfrac+icefrac

================================================================================

   New user specified fields

================================================================================
 New fields that are user specidied can be added as namelist variables
 by the user in the cpl namelist seq_flds_user using the namelist variable
 array cplflds_customs. The user specified new fields must follow the
 above naming convention.
 As an example, say you want to add a new state 'foo' that is passed
 from the land to the atm - you would do this as follows
    apos;seq_flds_user
       cplflds_custom = 'Sa_foo->a2x', 'Sa_foo->x2a'
    /
 This would add the field 'Sa_foo' to the character strings defining the
 attribute vectors a2x and x2a. It is assumed that code would need to be
 introduced in the atm and land components to deal with this new attribute
 vector field.
 Currently, the only way to add this is to edit $CASEROOT/user_nl_cpl
================================================================================

   Coupler fields use cases

================================================================================
 Previously, new fields that were needed to be passed between components
 for certain compsets were specified by cpp-variables. This has been
 modified to now be use cases. The use cases are specified in the
 namelist cpl_flds_inparm and are currently triggered by the xml
 variables CCSM_VOC, CCSM_BGC and GLC_NEC.
================================================================================
```

EXPERTS: How do I add a new user-defined component set?

Numerous component sets (i.e. compsets)[9] are provided "out-of-the-box" with CESM release. You can also call **create_newcase** giving it the -user_compset argument to point to your own customized component set name.

In CESM1.2, the component set definition file, `$CCSMROOT/scripts/ccsm_utils/Case.template/config_compsets.xml` has been redefined to be hierarchical. The following documents the rules involved for generating a compset from the hierarchy. The compononent set longname is given by the following notation:

```
TIME_ATM[%phys]_LND[%phys]_ICE[%phys]_OCN[%phys]_ROF[%phys]_GLC[%phys]_WAV[%phys][_BGC%

TIME = Time period (e.g. 2000, 20TR, RCP8...)
ATM  = [CAM4, CAM5, DATM, SATM, XATM]
LND  = [CLM40, CLM45, DLND, SLND, XLND]
ICE  = [CICE, DICE, SICE, SICE]
OCN  = [POP2, DOCN, SOCN, XOCN,AQUAP]
ROF  = [RTM, DROF, SROF, XROF]
GLC  = [CISM1, SGLC, XGLC]
WAV  = [SWAV, XWAV]
BGC  = optional BGC scenario
```

The optional %phys attributes specify submodes of the given system ALL the possible %phys choices for each component are listed with the -list command for create_newcase and also summarized below.

```
=========================================
Time period (first four characters)
=========================================
1850 => pre-industrial
2000 => present day
20TR => transient 1850 to 2000
5505 => transient 1955 to 2005
9205 => transient 1992 to 2005
RCP8 => transient RCP8.5 future scenario
RCP6 => transient RCP6.0 future scenario
RCP4 => transient RCP4.5 future scenario
RCP2 => transient RCP2.6 future scenario
NUKE => Nuclear winter hypothetical scenario (based on RCP4.5)
1996 => present day with conditions for solar minimum in 1996
AMIP => transient for "stand-alone" CAM (1979 startdate)
GEOS => GEOS5 metereology for "stand-alone" CAM

=========================================
CAM
=========================================
CAM4% => cam4 physics
CAM5% => cam5 physics
CAM[45]%WCCM => CAM WACCM with daily solar data and SPEs:
CAM[45]%WCMX => CAM WACCM-X:
CAM[45]%WCSC => CAM WACCM specified chemistry:
CAM[45]%WCBC => CAM WACCM with the stratospheric black carbon CARMA model:
CAM[45]%WCSF => CAM WACCM with sulfur chemistry and the sulfate CARMA model:
CAM[45]%FCHM => CAM super_fast_llnl chemistry:
CAM[45]%TMOZ => CAM trop_mozart chemistry:
CAM[45]%MOZM => CAM trop_mozart_mam3 chemistry:
CAM[45]%MOZS => CAM trop_mozart_soa chemistry:
CAM[45]%SMA3 => CAM trop_strat_mam3 chemistry:
CAM[45]%SMA7 => CAM trop_strat_mam7 chemistry:
CAM[45]%SSOA => CAM trop_strat_soa chemistry:
CAM[45]%RCO2 => CAM CO2 ramp:

=========================================
CLM
=========================================
note: [^_]* means match zero or more of any character BUT an underbar.
(in other words make sure there is NOT a underbar before the string afterwards)

CLM40            => clm4.0 Physics
CLM40%[^_]*SP    => clm4.0 Satellite phenology
CLM40%[^_]*CN    => clm4.0 Carbon Nitrogen
CLM40%[^_]*CNDV  => clm4.0 Carbon Nitrogen Dynamic Vegetation
CLM40%[^_]*CROP  => clm4.0 Prognostic crop
CLM40%[^_]*SNCR  => clm4.0 SNICAR radiative forcing calculation on

CLM45            => clm4.5 Physics
CLM45%[^_]*SP    => clm4.5 Satellite phenology
CLM45%[^_]*CN    => clm4.5 Carbon Nitrogen Biogeochemistry (BGC) (as in CLM4.0)
CLM45%[^_]*CNDV  => clm4.5 Carbon Nitrogen BGC with Dynamic Vegetation
CLM45%[^_]*BGC   => clm4.5 BGC (CN with vertically resolved soil BGC, based on Century
CLM45%[^_]*CROP  => clm4.5 Prognostic crop
CLM45%[^_]*VIC   => clm4.5 VIC hydrology
CLM40%[^_]*SNCR  => clm4.0 SNICAR radiative forcing calculation on
CLM45%[^_]*BGCDV => clm4.5 BGC (CN with vertically resolved soil BGC, based on Century

=========================================
CICE
=========================================
```

```
CICE      => prognostic cice
CICE%PRES => prescribed cice

========================================
POP2
========================================
POP2     => POP2 default
POP2%ECO => POP2/Ecosystem
POP2%DAR => Darwin marine ecosystem (not supported in community releases)

========================================
RTM
========================================
RTM       => default RTM model
RTM%FLOOD => RTM model with flood

========================================
CISM
========================================
CISM1  => cism1 (default, serial only)

========================================
DATM
========================================
DATM%QIA   => QIAN atm input data (1948-1972)
DATM%CRU   => CRUNCEP atm input data for (1901-2010)
DATM%S1850 => CPL history atm input data
DATM%1PT   => single point tower site atm input data
DATM%NYF   => COREv2 datm normal year forcing
DATM%IAF   => COREv2 datm interannual year forcing

========================================
DLND
========================================
DLND%NULL => dlnd_mode is NULL   , dlnd_sno_mode is NULL
DLND%SCPL => dlnd_mode is NULL   , dlnd_sno_mode is CPLHIST (used for TG)
DLND%LCPL => dlnd_mode is CPLHIST, dlnd_sno_mode is NULL

========================================
DROF
========================================
DROF%NYF  => COREv2 drof normal year forcing
DROF%IAF  => COREv2 drof interannual year forcing
DROF%NULL => null mode

========================================
DICE
========================================
DICE%SSMI => dice mode is ssmi
DICE%SIAF => dice mode is ssmi_iaf
DICE%PRES => dice mode is prescribed
DICE%COPY => dice mode is copy
DICE%NULL => dice mode is null

========================================
DOCN
========================================
DOCN%NULL => docn null mode
DOCN%SOM  => docn slab ocean mode
DOCN%DOM  => docn data mode
DOCN%US20 => docn us20 mode
DOCN%COPY => docn copy mode
```

There are two ways to create a customized user-defined component set. If the component set you want is not listed in the supported component sets[10], and you have no new optional %phys definitions for any of the components, then using the above

definitions you can create your own component set on the fly by using your own longname definition to create_newcase. As an example, the following will create a compset that is not currently supported out-of-the-box in CESM1.2.

```
> ./create_newcase -case mycompset \
  -user_compset 1850_CAM5_CLM45%CN_CICE_POP2_RTM_SGLC_SWAV \
  -res ne30_g16 \
  -mach yellowstone
```

If you want to create a component set that has new physics definitions, then the process is a bit more complicated. You will need to first edit `$CCSMROOT/scripts/ccsm_utils/Case.template/config_compset.xml` and fill in the appropriate sections specified by the string "USER_DEFINED section" as necessary. At that point, you can then call ./create_newcase as above with the -user_compset argument that is now customized to our requirements.

EXPERTS: How do I add a new user-defined grid?

Support for numerous out-of-the box model resolutions[11] accompany the CESM release. (In addition to the link above, you can also view a listing of supported "out-of-the-box" resolutions by running **create_newcase -l**.) In general, CESM grids are associated with a specific combination of atmosphere, land, land-ice, river-runoff and ocean/ice grids. The naming convention for these grids still only involves atmosphere, land, and ocean/ice grid specifications.

The most common resolutions have the atmosphere and land components on one grid and the ocean and ice on a second grid. The naming convention looks like *f19_g16*, where the f19 indicates that the atmosphere and land are on the 1.9x2.5 (finite volume dycore) grid while the g16 means the ocean and ice are on the gx1v6 one-degree displaced pole grid. While it is not supported, as of CESM1.1.1 does have the ability to run with the atmosphere and land also separated. The naming convention for these trigrid cases looks like *ne30_f19_g16*, where the ne30 means that the atmosphere is on the 30-element (spectral-element dycore) grid while the land is still on the finite volume grid and the ocean / ice are still on the gx1v6 grid. This document will outline how to set up the more complicated trigrid case, but will also highlight what steps can be skipped if the atmosphere and land do not need to be separated.

Note: This will be generalized in CESM1.2. TO DO

CESM provides completely *new support* for you to add your own specific component grid combinations. To achieve this, CESM has a new top level directory `$CCSMROOT/mapping/`. A brief list of the steps needed to add a new component grid to the model system follows. Again, this process can be simplified if the atmosphere and land are running on the same grid.

1. **Start with SCRIP grid files for atmosphere, land, and ocean.**

   You must first create or obtain SCRIP format grid files for the atmosphere, land and ocean grids. At present there is no supported functionality for creating the SCRIP format file, although that is planned for CESM1.2. (check)

2. **Build the `check_map` utility.**

   When you add new user-defined grid files, you will also need to generate a set of mapping files so the coupler can send data from a component on one grid to a component on another grid. There is an ESMF tool that tests the mapping file by comparing a mapping of a smooth function to its true value on the destination grid. We have tweaked this utility to test a suite of of smooth functions, as

well as ensure conservation (when the map is conservative). Before generating mapping functions it is *highly recommended* that you build this utility.

To build this tool, follow the instructions in `$CCSMROOT/mapping/check_maps/INSTALL`. As with many of the steps in this document, you will need to have the ESMF[12] toolkit installed. It is installed by default on most NCAR computers.

3. **Generate atm<->ocn, atm<->lnd, lnd<->rtm, and ocn->lnd mapping files.**

   Using the SCRIP grid files from step one, you must generate a set of conservative (area-averaged) and non-conservative (patch and bilinear) mapping files. You can do this by calling **gen_cesm_maps.sh** in `$CCSMROOT/tools/mapping/gen_mapping_files/`. This shell script generates all the mapping files needed by CESM (except rtm->ocn, which is discussed below). This script uses the ESMF offline weight generation utility[13], which you must build *prior* to running **gen_cesm_maps.sh**.

   The `README` file in the `gen_mapping_files/` directory contains details on how to run **gen_cesm_maps.sh** . The basic usage is

```
$ cd $CCSMROOT/mapping/gen_mapping_files
$ ./gen_cesm_maps.sh \
    --fileocn  <input SCRIP ocn_grid full pathname>  \
    --fileatm  <input SCRIP atm grid full pathname>  \
    --filelnd  <input SCRIP lnd grid full pathname>  \
    --filertm  <input SCRIP rtm grid full pathname>  \
    --nameocn  <ocnname in output mapping file> \
    --nameatm  <atmname in output mapping file> \
    --namelnd  <lndname in output mapping file> \
    --namertm  <rtmname in output mapping file>
```

   This command will generate the following mapping files:

```
map_atmname_TO_ocnname_aave.yymmdd.nc
map_atmname_TO_ocnname_blin.yymmdd.nc
map_atmname_TO_ocnname_patc.yymmdd.nc
map_ocnname_TO_atmname_aave.yymmdd.nc
map_ocnname_TO_atmname_blin.yymmdd.nc
map_atmname_TO_lndname_aave.yymmdd.nc
map_atmname_TO_lndname_blin.yymmdd.nc
map_lndname_TO_atmname_aave.yymmdd.nc
map_ocnname_TO_lndname_aave.yymmdd.nc
map_lndname_TO_rtmname_aave.yymmdd.nc
map_rtmname_TO_lndname_aave.yymmdd.nc
```

   Notes:

   a. You do not need to specify all four grids. For example, if you are running with the atmosphere and land on the same grid, then you do not need to specify the land grid (and atm<->rtm maps will be generated). If you also omit the runoff grid, then only the 5 atm<->ocn maps will be generated.

   b. ESMF_RegridWeightGen runs in parallel, and the `gen_cesm_maps.sh` script has been written to run on yellowstone. To run on any other machine, you may need to add some environment variables to `$CCSMROOT/mapping/gen_mapping_files/gen_ESMF_mapping_file/create_ESMF_map.sh` -- search for *hostname* to see where to edit the file.

   Example (run on Nov 5, 2012):

```
$ ./gen_cesm_maps.sh \
     -focn /CESM/cseg/mapping/grids/gx3v7_120309.nc -nocn g37 \
     -fatm /CESM/cseg/mapping/grids/ne16np4_110512_pentagons.nc -natm ne16np4 \
     -frtm /CESM/cseg/mapping/grids/r05_nomask_070925.nc -nrtm r05
```

   Results in the following files

```
$ ls -1 map*
map_g37_TO_ne16np4_aave.121105.nc
map_g37_TO_ne16np4_blin.121105.nc
map_ne16np4_TO_g37_aave.121105.nc
map_ne16np4_TO_g37_blin.121105.nc
map_ne16np4_TO_g37_patc.121105.nc
map_ne16np4_TO_r05_aave.121105.nc
map_r05_TO_ne16np4_aave.121105.nc
```

4. **Generate atmosphere, land and ocean / ice domain files.**

   Using the conservative ocean to land and ocean to atmosphere mapping files created in the previous step, you can create domain files for the atmosphere, land, and ocean; these are basically grid files with consistent masks and fractions. You make these files by calling **gen_domain** in `$CCSMROOT/mapping/gen_domain_files`.

   The `INSTALL` file in the `gen_domain_files/` directory contains details on how to build the **gen_domain** executable. After you have built it, the `README` in the same directory contains details on how to use the tool. The basic usage is:

```
$ ./gen_domain -m ../gen_mapping_files/map_ocnname_TO_lndname_aave.yymmdd.nc \
               -o ocnname -l lndname
$ ./gen_domain -m ../gen_mapping_files/map_ocnname_TO_atmname_aave.yymmdd.nc \
               -o ocnname -l atmname
```

   These commands will generate the following domain files:

```
domain.lnd.lndname_ocnname.yymmdd.nc
domain.ocn.lndname_ocnname.yymmdd.nc
domain.lnd.atmname_ocnname.yymmdd.nc
domain.ocn.atmname_ocnname.yymmdd.nc
domain.ocn.ocnname.yymmdd.nc
```

   Notes:

   a. If you are running with the atmosphere and land components on the same grid, you only need to execute **gen_domain** once.

   b. The input atmosphere grid is assumed to be unmasked (global). Land cells whose fraction is zero will have land mask = 0.

   c. If the ocean and land grids *are identical* then the mapping file will simply be unity and the land fraction will be one minus the ocean fraction.

5. **If you are adding a new ocn or rtm grid, create a new rtm->ocn mapping file.** (Otherwise you can skip this step.)

   The process for mapping from the runoff grid to the ocean grid is currently undergoing many changes. At this time, if you are running with a new ocean or runoff grid, please contact Michael Levy (mlevy_AT_ucar_DOT_edu) for assistance. If you are running with standard ocean and runoff grids, the mapping file should already exist and you do not need to generate it.

6. **If you are adding a new new lnd grid, create a new CLM surface dataset.** (Otherwise you can skip this step.)

   a. Generate mapping files for CLM surface dataset (since this is a non-standard grid).

```
$ cd $CCSMROOT/models/lnd/clm/tools/mkmapdata
$ ./mkmapdata.sh --gridfile <lnd SCRIP grid file> \
                 --res <atm resolution name> \
                 --gridtype global
```

   b. Generate CLM surface dataset. Below is an example for a current day surface dataset (model year 2000).

```
$ cd  $CCSMROOT/models/lnd/clm/tools/mksurfdata_map
$ ./mksurfdata.pl -res usrspec -usr_gname <atm resolution name> \
                  -usr_gdate yymmdd -y 2000
```

7. **Create grid file needed for create_newcase.**

   The next step is to create a file - call it `mygrid.xml` - with all the grid and domain information. Assuming the domain files that were generated earlier are in `$DOMAIN_FILE_LOC`, the contents of this file should be

```
<?xml version="1.0"?>
<config_horiz_grid>
<horiz_grid GLOB_GRID="atmgrid" nx="[size of atmgrid]" ny="[size of atmgrid]" />
<horiz_grid GLOB_GRID="lndgrid" nx="[size of lndgrid]" ny="[size of lndgrid]" />
<horiz_grid GLOB_GRID="ocngrid" nx="[size of ocngrid]" ny="[size of ocngrid]" />
<horiz_grid GRID="atmgrid_lndgrid_ocngrid" SHORTNAME="atm_lnd_ocn"
            ATM_GRID="atmgrid" LND_GRID="lndgrid" OCN_GRID="ocngrid" ICE_GRID="ocn
            ATM_NCPL="48" OCN_NCPL="1"
            ATM_DOMAIN_FILE="domain.lnd.atmgrid_ocngrid.$YYYYMMDD.nc"
            LND_DOMAIN_FILE="domain.lnd.lndgrid_ocngrid.$YYYYMMDD.nc"
            ICE_DOMAIN_FILE="domain.ocn.ocngrid.$YYYYMMDD.nc"
            OCN_DOMAIN_FILE="domain.ocn.ocngrid.$YYYYMMDD.nc"
            ATM_DOMAIN_PATH="$DOMAIN_FILE_LOC"
            LND_DOMAIN_PATH="$DOMAIN_FILE_LOC"
            ICE_DOMAIN_PATH="$DOMAIN_FILE_LOC"
            OCN_DOMAIN_PATH="$DOMAIN_FILE_LOC"
            DESC="Some new trigrid setup"
/>
</config_horiz_grid>
```

   Where you only need the `GLOB_GRID` information for grids that are not already included in the model. For unstructured grids, `nx` should be the number of grid cells and `ny` should be 1; for structured grids, they should be the dimensions of the grid.

8. **Create user_nl_cpl contents for new mapping files.**

   One of the many input files generated for the coupler is `$RUNDIR/seq_maps.rc`, which contains a list of mapping files. Using an f09_g16 run on yellowstone as an example, the file will contain the following (for brevity, some lines have been cut):

```
 atm2ocnFmapname: '/glade/proj3/cseg/inputdata/cpl/cpl6/map_fv0.9x1.25_to_gx1v6_aa
 atm2ocnSmapname: '/glade/proj3/cseg/inputdata/cpl/cpl6/map_fv0.9x1.25_to_gx1v6_bi
 atm2ocnVmapname: '/glade/proj3/cseg/inputdata/cpl/cpl6/map_fv0.9x1.25_to_gx1v6_bi
 lnd2atmFmapname: 'idmap'
 lnd2atmSmapname: 'idmap'
 lnd2rofFmapname: '/glade/proj3/cseg/inputdata/lnd/clm2/mappingdata/maps/0.9x1.25/
 lnd2rofFmaptype: 'X'
 ocn2atmFmapname: '/glade/proj3/cseg/inputdata/cpl/cpl6/map_gx1v6_to_fv0.9x1.25_aa
 ocn2atmSmapname: '/glade/proj3/cseg/inputdata/cpl/cpl6/map_gx1v6_to_fv0.9x1.25_aa
```

   This file is created when you build the model namelists, and the default values are based on the grids specified when you created the case. The model only knows what default values to use for the out-of-the-box resolutions, so you must specify what maps you have created by appending them to `$CASE/user_nl_cpl`. If, for example, we've introduced a new atmosphere / land grid with a shortname newatm and created all the necessary mapping files in `$MAPPING_FILE_LOC`, then to create a newatm_g16 run we would need to add the following to `$CASE/user_nl_cpl`:

```
atm2ocnFmapname='$MAPPING_FILE_LOC/map_newatm_TO_gx1v6_aave.121105.nc'
atm2ocnSmapname='$MAPPING_FILE_LOC/map_newatm_TO_gx1v6_blin.121105.nc'
atm2ocnVmapname='$MAPPING_FILE_LOC/map_newatm_TO_gx1v6_patc.121105.nc'
```

```
ocn2atmFmapname='$MAPPING_FILE_LOC/map_gx1v6_TO_newatm_aave.121105.nc'
ocn2atmSmapname='$MAPPING_FILE_LOC/map_gx1v6_TO_newatm_aave.121105.nc'
lnd2rofFmapname='$MAPPING_FILE_LOC/map_newatm_TO_r05_aave.121105.nc'
rof2lndFmapname='$MAPPING_FILE_LOC/map_r05_TO_newatm_aave.121105.nc'
```

After running `$CASE/preview_namelists` these changes will be reflected in `$RUNDIR/seq_maps.rc`.

9. **Test new grid.**

   Below assume that the new grid is an atmosphere grid.

```
Test the new grid with all data components.
(write an example)
Test the new grid with CAM(newgrid), CLM(newgrid), DOCN(gx1v6), DICE(gx1v6)
(write an example)
```

EXPERTS: How do I carry out data assimilation using CAM and DART?

Ensemble Kalman filter data assimilation (DA) can now be conducted within the software framework of CESM. This form of DA uses the multi-instance capability of CESM in which CESM advances an ensemble of model states of one or more CESM components forward to the same forecast time, when observations are available. Then the ensemble of forecast model states is passed to the Data Assimilation Research Testbed (DART), where each state is adjusted toward the observations which are available at that time. For details of this process see an introduction in BAMS (2009)[14] and/or the DART home page[15]. DART then passes the ensemble of adjusted model states back to CESM to be used as initial conditions for the next forecast.

The references above describe the many uses of ensemble data assimilation, which include:

- generation of analyses (blends of model forecast and observations which are a better description of the physical system than either by itself),
- model development testing against actual observations (as opposed to other analyses),
- sensitivity analysis between model variables of interest in a particular synoptic situation,
- variability studies using the ensemble of equally valid model states, observation system simulation experiments (OSSEs).

This use case outlines assimilation for a CAM (F comp set) build only. Assimilation is possible with the ocean component (B comp sets), and experimental assimilations with the land component (I comp sets) have been conducted. Additional use case descriptions will be added to cover those and any future evolution of the CESM+DART software. This use case assumes that the user is familiar with setting up and using CESM, and is willing to learn how to set up and use DART in the CESM context. There is no simple example which users can grab and run, because understanding what is being run is crucial to success and there are many choices to be made.

The major steps of assimilating observations into CAM follow.

1. Download DART[16]. DART relieves researchers of the need to develop data assimilation capabilities, but familiarity with data assimilation and the DART facility is required in order to use it productively. This can be gained through the DART tutorial[17].

2. Build the DART executables [18] for a simple model to check that DART has been installed correctly.
3. Build the DART executables for CAM, following a similar procedure to 2.
4. The script .../DART/models/cam/CESM_setup.csh builds a CAM which combines the user's desired features and DART's required features. The characteristics of the CAM and assimilation set in CESM_setup.csh are:
   - locations of the build, run, and archive directories,
   - features of the $CASE to be built,
   - locations of input files, including the initial ensemble of CAM (and CLM and CICE) states
   - date and timing characteristics of the assimilation,
   - machine and resource characteristics.

   ```
   - Copy CESM_setup.csh to the directory where the user wants to build CAM.
   - Edit that CESM_setup.csh to set most of the assimilation parameters.
   - Run CESM_setup.csh.
   ```

5. Set the rest of the assimilation parameters:

   ```
   - cd to $CASE
   - Edit input.nml to set other characteristics of the assimilation.
     For details see the online help pages[19] or the html in the user's $DART/filter/
   - Edit assimilate.csh to set the location of the observations to be assimilated.
     Sets of real observations are available for use, or synthetic observations[20]
     can be created using the user's model.
   ```

6. Submit the job using $CASE.submit in the $CASEROOT directory.
7. Output from the assimilation is handled by the CESM archiver(s), which has been modified to handle DART output. Output appears in a new short-term archive directory .../archive/.../dart/hist. The 3 files created at each assimilation time are
   - Prior_Diag.YYYY-MM-DD-SSSSS.nc: the ensemble mean, spread, members (optionally), and 'inflation' fields from before the assimilation (at the end of the forecast).
   - Posterior_Diag.YYYY-MM-DD-SSSSS.nc: same as Prior, but from after the assimilation.
   - obs_seq.YYYY-MM-DD-SSSSS.final: the actual observations assimilated and the ensemble members estimates of those observations.

   ```
   The obs_seq.final files are usually processed by the obs_diag[21]
   program in DART (.../DART/diagnostics/threed_sphere/obs_diag.f90),
   and the resulting NetCDF files are usually processed with Matlab scripts
   included in DART (or similar).  Little knowledge of Matlab is needed to use them.
   The Prior and Posterior files can be examined with any NetCDF viewing tool.
   ```

EXPERTS: How do I add a new CESM model component?

The following provides a very general overview of what you need to do to add a new atm, lnd, ocn, ice, glc or rof component to CESM.

1. You must support init, run, and final top level interfaces.
2. You must "send" the component grid and decomposition at initialization.
3. You must pack and unpack coupling fields to/from interface datatypes.

4. You must integrate forward a fixed amount of time and confirm that your model is in sync with the driver clock.
5. You must provide/use "expected" scalar information as needed. - provide present/prognostic flags at initialization - provide "nextsw_cday" if atm component, use nextsw_cday if surface model - use mpicom - use stop and restart information - use inst_name, inst_index, inst_suffix (for cesm[version]) - use other infodata information as needed (ie. starttype, case_name, configuration settings like aqua_planet, orbital settings)
6. Use I/O unit manager in CESM
7. You must meet filename conventions for history, restart, and log files

There are some component to component variations in the interfaces and in the provide/use of scalar data, so it's best to follow another component of the same flavor. The top level interface will be a fortran file called ***_comp_mct.F90 where *** is atm, ocn, ice, or lnd. That file exists in all components. Below is a generic summary of what is going on using the atm component as an example. Other components are very close.

1. You must support init, run, and final top level interfaces. The interfaces must follow the naming convention and argument types exactly. These interfaces must be in a file called atm_comp_mct.F90 and the module must be called atm_comp_mod. The driver will access the component model only through the init, run, and final interfaces.

```
module atm_comp_mct

public :: atm_init_mct
public :: atm_run_mct
public :: atm_final_mct

subroutine atm_init_mct( EClock, cdata_a, x2a_a, a2x_a, NLFilename )
type(ESMF_Clock),intent(in)                 :: EClock
type(seq_cdata), intent(inout)              :: cdata_a
type(mct_aVect), intent(inout)              :: x2a_a
type(mct_aVect), intent(inout)              :: a2x_a
character(len=*), optional,   intent(IN)    :: NLFilename ! Namelist filenam

subroutine atm_run_mct( EClock, cdata_a, x2a_a, a2x_a)
type(ESMF_Clock)            ,intent(in)    :: EClock
type(seq_cdata)             ,intent(inout) :: cdata_a
type(mct_aVect)             ,intent(inout) :: x2a_a
type(mct_aVect)             ,intent(inout) :: a2x_a

subroutine atm_final_mct( )
```

2. You must "send" the component grid and decomp at initialization The cdata datatype contains data for a grid and decomp. The decomp is an mct gsmap and the grid is an general grid. To access these data type from the init method, do the following.

```
type(mct_gsMap), pointer   :: gsMap_atm
type(mct_gGrid), pointer   :: dom_a

call seq_cdata_setptrs(cdata_a, gsMap=gsMap_atm, dom=dom_a)

! call an mct_gsmap_init method and specify the global index
!   of each local gridcell
! call an mct_gGrid_init method and fill the lon/lat/area/mask/frac
!   arrays
```

3. You must pack and unpack coupling fields to/from interface datatypes The fields coupling datatypes are mct attribute vectors. x2a contains the coupler->atm fields. a2x contains the atm->coupler fields. This datatype must be initial-

ized by the component in the init method. To do that use the fields list provided by seq_flds_mod and initialize the gsmap first. lsize below is the local number of gridcells on the processor. the mct_aVect_init calls below allocate arrays in the attribute vector to store the appropriate number of fields of appropriate local size.

```
use seq_flds_mod

lsize = mct_gsMap_lsize(gsMap_atm, mpicom_atm)
call mct_aVect_init(a2x_a, rList=seq_flds_a2x_fields, lsize=lsize)
call mct_aVect_zero(a2x_a)
call mct_aVect_init(x2a_a, rList=seq_flds_x2a_fields, lsize=lsize)
call mct_aVect_zero(x2a_a)
```

To pack the data, it's easiest just to write directly into the arrays inside the attribute vector in the following manner,

```
integer :: index_a2x_Sa_pslv         ! sea level atm pressure

index_a2x_Sa_pslv  = mct_avect_indexra(a2x_a,'Sa_pslv')

do i=is,ie
   a2x_a%rAttr(index_a2x_Sa_pslv ,i) = psl(i)
   a2x_a%rAttr(index_a2x_Sa_z    ,i) = zbot(i)
   a2x_a%rAttr(index_a2x_Sa_u    ,i) = ubot(i)
   a2x_a%rAttr(index_a2x_Sa_v    ,i) = vbot(i)
enddo
```

To unpack, basically do the same thing in the opposite direction.

```
integer :: index_x2a_Sx_t            ! surface temperature

index_x2a_Sx_t  = mct_avect_indexra(x2a_a,'Sx_t')

do i=is,ie
   ts(i) = x2a_a%rAttr(index_x2a_Sx_t ,i)
enddo
```

The attribute vectors store only the "local" data and you basically just need to copy data from the model datatype to the coupling MCT (or ESMF) datatype.

4. You must integrate forward a fixed amount of time and confirm that your model is in sync with the driver clock. You can access clock information from the EClock passed through the coupling interfaces and the EClockGetData method. The approach to sync the model and driver clock is very model specific. In some cases, a model may just get the current time from the EClock and use it. That probably only happens for models that don't advance in time (like maybe a data model). In other cases, model clocks may be initialized based on the EClock data at initialization and then the model time and driver time are regularly compared for consistency. Another approach is to use the "dt" provided by the EClock to advance a model "dt" seconds. Some examples are provided.

```
type(ESMF_Clock)            ,intent(in)    :: EClock

! EClock initialization data
call seq_timemgr_EClockGetData(EClock, &
   start_ymd=start_ymd, start_tod=start_tod, &
   ref_ymd=ref_ymd, ref_tod=ref_tod,         &
   stop_ymd=stop_ymd, stop_tod=stop_tod,     &
   calendar=calendar )

! EClock dt
call seq_timemgr_EClockGetData(Eclock,dtime=atm_cpl_dt)

! EClock current time
call seq_timemgr_EClockGetData(EClock,curr_ymd=ymd_sync,curr_tod=tod_sync, &
   curr_yr=yr_sync,curr_mon=mon_sync,curr_day=day_sync)
```

```
! Check synchronization
ymd=model_ymd
tod=model_tod
if (.not. seq_timemgr_EClockDateInSync( EClock, ymd, tod)) then
   write(*,*) "Clocks not in sync"
   call shr_sys_abort()
endif
```

5. You must provide/use "expected" scalar information as needed. Each component varies a bit wrt what's provided and used.
   - Provide present/prognostic flags at initialization. The logical flag present means the component provides data. The logical flag prognostic means the component uses data. Stub models set both to false. Data models generally set present to true and prognostic to false, except in cases where a data model needs some coupling data for some internal computations (e.g. DOCN-SOM). Active models generally set both flags to true.

     ```
     call seq_infodata_PutData(infodata, atm_present=.true.)
     call seq_infodata_PutData(infodata, atm_prognostic=.true.)
     ```

   - Provide "nextsw_cday" if you are adding an atm component. Use nextsw_cday if you are adding a surface model. The real variable nextsw_cday is the time of the next atm radiation calculation if it occurs at the next coupling period. If there is no radiation calculation on the next timestep, this should be set to -1. This allows surface albedos and the atm radiation calculation to stay synced up. The surface models use the nextsw_day field and compute albedos based on that time.

     ```
     call seq_infodata_PutData( infodata, nextsw_cday=nextsw_cday )
     ```

   - Use the mpi communicator, mpicom, which is provided by the driver to the component. This must be used for all internal model communication. The data is provided in the cdata datatype.

     ```
     call seq_cdata_setptrs(cdata_a, mpicom=mpicom_atm)
     ```

   - Use stop and restart information provided by the driver. The driver will tell the component if this is the last coupling period and/or if a restart is required at the end of this coupling period. The component should listen to both these flags.

     ```
     stop_now = seq_timemgr_StopAlarmIsOn(EClock)
     restart_now = seq_timemgr_RestartAlarmIsOn(EClock)
     ```

   - Use inst_name, inst_index, inst_suffix needed for for the multiple instance capability. As a starting point, the following provides standard code that can be added:
   -

     ```
     integer(IN)   :: COMPID                ! mct comp id
     integer       :: inst_index            ! number of current instance (ie.
     character(len=16) :: inst_name         ! fullname of current instance (i
     character(len=16) :: inst_suffix       ! char string associated with ins

     call seq_cdata_setptrs(cdata, ID=COMPID)
     inst_name   = seq_comm_name(COMPID)
     inst_index  = seq_comm_inst(COMPID)
     inst_suffix = seq_comm_suffix(COMPID)
     ```

   - Use other infodata information as needed (ie. starttype, case_name, configuration settings like aqua_planet, orbital settings)

6. Use the I/O unit manager in CESM. To avoid conflicts in I/O unit numbers between components, models should call the shr_file_getUnit and shr_file_freeUnit methods to acquire and release available unit numbers.

   ```
   nunit = shr_file_getUnit()
   ```

```
open(nunit,file='xyz')
read(nunit,*) xyz
close(nunit)
call shr_file_freeUnit(nunit)
```

7. Meet filename conventions for history, restart, and log files. There are specific filename conventions for CESM. In particular, all history files should be CF compliant netcdf. This format is also recommended for restart files. The format is something like

```
$CASE.atm.ha.2001-01.nc         ! history
$CASE.atm.r.2001-01-00000.nc    ! restart
```

The log files are set by a shared method. In particular, models should do something like

```
!--- open log file ---
if (my_task == master_task) then
   logUnit = shr_file_getUnit()
   call shr_file_setIO('atm_modelio.nml',logUnit)
else
   logUnit = 6
endif
```

That logunit value should then be used by all processors in the model to write "stdout" messages. shr_file_setIO associates the logUnit number with a unique log filename for the case. unit 6 is used on non-root processors and information written from those processors goes to a stdout file which is machine dependent.

Notes

1. ../modelnl/env_build.html#build_def
2. ../modelnl/env_run.html
3. http://www.earthsystemmodeling.org/download/index.shtml
4. http://www.cesm.ucar.edu/models/cesm1.2/cesm/doc/modelnl/env_run.html#run_pio
5. http://www.cesm.ucar.edu/models/pio/
6. ../modelnl/nl_cam.html
7. ../modelnl/nl_cam.html
8. ../../../cpl7/doc/book1.html
9. ../modelnl/compsets.html
10. ../modelnl/compsets.html
11. ../modelnl/grid.html
12. http://www.earthsystemmodeling.org
13. http://www.earthsystemmodeling.org
14. http://journals.ametsoc.org/doi/abs/10.1175/2009BAMS2618.1
15. http://www.image.ucar.edu/DAReS/DART/
16. http://www.image.ucar.edu/DAReS/DART
17. http://www.image.ucar.edu/DAReS/DART
18. http://www.image.ucar.edu/DAReS/DART
19. https://proxy.subversion.ucar.edu/DAReS/DART/trunk/filter/filter.html#GettingStarted
20. http://www.image.ucar.edu/DAReS/DART/DART_Observations.php#obs_synthetic
21. http://subversion.ucar.edu/DAReS/DART/trunk/diagnostics/threed_sphere/obs_diag.html

Chapter 7. CESM Testing

Testing overview

CESM1.2 is accompanied by updated utilities that support automated testing of the model. In general, these should be used only after the model has been ported to the target machine (see Chapter 5). The tools are **create_production_test** and $**create_test**.

The **create_production_test** tool is executed from your $CASEROOT, and tests your case's ability to be restarted in a bit-for-bit fashion in a separate directory.

New perl-based $**create_test** and $**query_tests** utilities are executed from $CCSMROOT/scripts and allow you to quickly determine what tests are available as well as set up and run one of numerous supported tests or create any one of numerous test suites that are used by the CESM developers for testing the model.

Using create_production_test

In general, after configuring and testing a case and before starting a long production job based on that case, it's important to verify that the model restarts exactly. This is a standard requirement of the system and will help demonstrate stability of the configuration technically. The tool create_production_test is located in the case directory, and it sets up an ERT two month exact restart test in the same directory as the current case. To use it, do the following

```
> cd $CASEROOT
> ./create_production_test
> cd ../$CASE_ERT
> ./$CASE_ERT.test_build
> ./$CASE_ERT.submit

Check your test results. A successful test produces "PASS" as
the first word in the file, $CASE_ERT/TestStatus
```

If the test fails, see the Section called *Debugging Tests That Fail* for test debugging guidance.

Using query_tests

In CESM1.2, automated regression and system tests are now found in a single xml-based file in $CCSMROOT&/scripts/ccsm_utils/Testlistxml/testlist.xml. The new utility **query_tests** allows you to quickly and easily query the different CESM test categories that are supported as well as what tests are available for different grids, compsets, machines and compilers. You should use this command to become familiar with the latest CESM testing capabilities before you utilize **create_test**. You should first use the -h option in calling **query_tests** to document its input options. **query_tests** can be called with the following arguments:

```
query_tests \
   -list name
        name can be [compsets,grids,compilers,machines,categories,tests] \
   -compset name
       limit selection to target compset name \
   -category name
       limit selection to target category name \
   -machine name
       limit selection to target machine name \
   -compiler name
```

```
        limit selection to target compiler name \
    -grid name
        limit selection to target grid name match \
    -outputlist
        outputs the found tests to the old-style text test lists. \
    -outputxml
        outputs the found tests to an xml test list.
```

As an example to see all the tests that are supported for the compset B1850C5CN, call $**query_tests** as follows

```
./query_tests -compset B1850C5CN
```

And the following output will appear

```
Compset                        TestName  Grid       Machine_compiler      Category
================================================================================
B1850C5CN (B_1850_CAM5_CN)     ERI       f19_g16    hopper_pgi            prerelease
B1850C5CN (B_1850_CAM5_CN)     ERI       f19_g16    intrepid_ibm          prerelease
B1850C5CN (B_1850_CAM5_CN)     ERI       ne30_g16   hopper_pgi            prebeta
B1850C5CN (B_1850_CAM5_CN)     ERI       ne30_g16   intrepid_ibm          prebeta
B1850C5CN (B_1850_CAM5_CN)     ERS       ne30_g16   yellowstone_pgi       prebeta
B1850C5CN (B_1850_CAM5_CN)     ERS       ne30_g16   yellowstone_gnu       prebeta
B1850C5CN (B_1850_CAM5_CN)     ERS       ne30_g16   yellowstone_intel     prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   edison_intel          prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   mira_ibm              prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   titan_pgi             prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   yellowstone_gnu       prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   yellowstone_pgi       prebeta
B1850C5CN (B_1850_CAM5_CN)     PFS       ne30_g16   yellowstone_intel     prebeta
```

To find all the tests that are configured to run on Yellowstone, run:

```
> ./query_tests -mach yellowstone
```

The output will show the compset, test name, grid, machine / compiler combinaton, test category, and and optional namelist directory. (used for specifying custom namelists for tests)

To find all the tests configured to run on titan, using the PGI compiler, and using the 'prebeta' category, run:

```
> ./query_tests -mach titan -compiler pgi -category prebeta
```

To find all the tests using the B1850 compset, run:

```
> ./query_tests -compset B1850
```

If one wanted a new test list based on a particular query, one can use the -outputxml option to get the query output in XML format:

```
> ./query_tests -category aux_clm -outputxml > aux_clm.xml
```

The above command will give one all of the 'aux_clm' tests in the file 'aux_clm.xml'. Then, this test list can be used by create_test to run the aux_clm tests using the -xml_list option.

Using create_test

The new $**create_test** tool is located in the `$CCSMROOT/scripts/` directory and can be used to set up entire CESM test suites, as well as single standalone test cases. To see the list of test cases, and to view the available script options, execute create_test -help or create_test without any arguments. Creating a single test looks like:

```
> cd $CCSMROOT/scripts
> ./create_test -testname ERS.f19_g16.X.yellowstone_intel -testid t01
> cd ERS.f19_g16.X.yellowstone_intel.t01
> ERS.f19_g16.X.yellowstone_intel.t01.test_build
./ERS.f19_g16.X.yellowstone_intel.t01.submit
Check your test results. A successful test produces "PASS" as
  the first word in the file TestStatus
```

The above commands set up an exact restart test (ERS) at the 1.9x2.5_gx1v6 resolution using a dead model component set (X) for the machine yellowstone. The testid provides a unique tag for the test in case it needs to be rerun (i.e. using -testid t02).

As an example, to create an entire suite of tests on yellowstone for the 'prebeta' test category, do the following:

```
> cd $CCSMROOT/scripts
> ./create_test \
     -xml_mach yellowstone \
     -xml_compiler intel \
     -xml_category prebeta \
     -testid alpha01a \
     -testroot /glade/scratch/$USER/alpha01a
```

The above command will run all of the 'prebeta' tests in the ccsm_utils/Testlistxml/testlist.xml file that are configured for the category prebeta using the Intel compiler on yellowstone. This is almost identical to the development testing that CSEG carries out on various machines. In addition to creating the suite of tests in your chosen test root, $**create_test** will also copy several helper scripts into the testroot directory. The first will be named cs.status.$testid.$machine. When run, this script will output the current status of all the tests. The second will be named cs.submit.$testid.$machine. This script is automatically run when all the test cases are created, and it builds and submits the suite of tests for you.

Some things to note about CESM tests:

- For usage information about the create_test tool, run "create_test -help".
- Test results are output in the TestStatus file. The TestStatus.out file provides additional details of the test run, which may aid in debugging failed tests.
- At this time, tests are not always easily re-runnable from an existing test directory. Rather than rerun a previous test case, it's recommended to set up a clean test case, (i.e. create one with a new testid)
- Tests are built using the .test_build script. This is different from regular production cases which are built using the .build script. Some tests require more than one executable, thus the .test_build script builds all required executables upfront interactively.
- The costs of tests vary widely. Some are short and some are long.
- If a test fails, see the Section called *Debugging Tests That Fail*for debugging assistance.
- There are -compare, -generate, and -baselineroot options for the create_test tool that support regression testing. These tools allow one to accomplish several goals:

- -generate will save log files as well as coupler history NetCDF files in the baseline-root under the current case name. Later tests will compare their coupler history files against these baselines to check for numerical differences.
- -compare will compare the current tag's tests against a previous tag's results, again for numerical accuracy.
- -baselineroot simply specifies where you would like your baseline files to be stored. By default, the test system will choose the configured baseline root for your machine.
- There are extra test options that can be added to the test such as _D, _E, or _P*. These are described in more detail in the create_test -help output.
- There is also a new option to create_test, -nlcompareonly. This allows one to create a suite of Smoke Build Namelist tests. These tests aren't compiled or run, the test cases are simply generated. These are useful in that you can create a suite for a previous CESM tag, then compare the current CESM tag's generated namelists against the previous tag. This can potentially spot hard-to-find answer-changing errors, and/or unintended changes in development.

The test status results have the following meaning

Test Result	Description
BFAIL	compare test couldn't find the baseline directory for the testcase
BUILD	build succeeded, test not submitted
CFAIL	env variable or build error
CHECK	manual review of data is required
ERROR	test checker failed, test may or may not have passed
FAIL	test failed
GEN	test has been generated
PASS	test passed
PEND	test has been submitted
RUN	test is currently running OR it hung, timed out, exceeded its allotted walltime, or exited abnormally
SFAIL	generation of test failed in scripts
TFAIL	test setup error
UNDEF	undefined result

The following tests are available at the time of writing:

Test	Description
SMS	smoke test
ERS	exact restart from startup, default 6 days initial + 5 days restart
ERB	branch/exact restart test
ERH	hybrid/exact restart test
ERI	hybrid/branch/exact restart test

Test	Description
ERT	exact restart from startup, default 2 months + 1 month
SEQ	sequencing bit-for-bit test
PEA	single processor testing with mpi and mpi-serial
PEM	pe counts mpi bit-for-bit test
PET	pe counts mpi/openmp bit-for-bit test
CME	compare mct and esmf interfaces test
NCK	single vs multi instance validation test
SBN	smoke build namelist test

Debugging Tests That Fail

This section describes what steps can be taken to try to identify why a test failed. The primary information associated with reviewing and debugging a run can be found in the Section called *Troubleshooting runtime problems* in Chapter 8.

First, verify that a test case is no longer in the batch queue. If that's the case, then review the possible test results and compare that to the result in the TestStatus file. Next, review the TestStatus.out file to see if there is any additional information about what the test did. Finally, go to the troubleshooting section and work through the various log files.

Finally, there are a couple other things to mention. If the TestStatus file contains "RUN" but the job is no longer in the queue, it's likely that the job either timed out because it exceeded its specified wall clock time, or the job hung or exited abnormally due to some run-time error. Check the batch log files to see if the job was killed due to a time limit, and if it was, increase the time limit and either resubmit the job, or generate a new test case and update the time limit before submitting it.

Also, a test case can fail because either the job didn't run properly or because the test conditions (i.e. exact restart) weren't met. Try to determine whether the test failed because the run failed, or because the test did not meet the test conditions. If a test is failing early in a run, it's usually best to set up a standalone case with the same configuration in order to debug problems. If the test is running fine, but the test conditions are not being met (i.e. exact restart), then that requires debugging of the model in the context of the test conditions.

Not all tests will pass for all model configurations. For more information, please check the known problems page for this release to find out which machines have problems with which compsets and/or resolutions.

- All models are bit-for-bit reproducible on different processor counts EXCEPT for POP2 and CICE diagnostics. The coupler is not bit-for-bit reproducible out of the box. The BFBFLAG must be set to TRUE in the `env_run.xml` file for the coupler to be bit-for-bit reproducible. If you have a configuration where you expect bit-for-bit reproducibility when you change the processor count AND you want to validate this, then the BFBFLAG must be set to TRUE in the env_run.xml file if the coupler is to meet this condition. The main purpose of the BFBFLAG is to enforce a specific order of operations in the mapping implementation. This constraint can impact mapping performance so it is recommended that the BFBFLAG be set to FALSE in production. Also note that the CESM system is fully bit-for-bit reproducible when rerunning the same configuration on the same processor count. The BFBFLAG is only required when trying to reproduce answers when changing processor counts.

- Some of the active components cannot run with the mpi serial library. This library takes the place of mpi calls when the model is running on one processors and MPI is not available or not desirable. The mpi serial library is invoked by setting the xml variable MPILIB to mpi-serial in `env_build.xml`. An effort is underway to extend the mpi serial library to support all components' usage of the mpi library with this standalone implementation. Also NOT all machines/platforms are set up to enable setting MPILIB to mpi-serial. See the summary of supported machines[1] for details. To use the mpi serial feature on your machine, you also need to make changes in the config_compilers.xml and config_machines.xml files for that machine. The best way to do this is to use a machine where MPILIB can be set to mpi-serial and look at the type of changes needed to make it work. Those same changes will need to be introduced for your machine. For the Macros file this includes the name of the compiler, possibly options to the compiler, and the settings of the MPI library and include path. For the config_machines.xml file you may want/need to modify the setting of MPICH_PATH. There also maybe many settings of MPI specific environment variables that don't matter when the mpiserial setting is used.

Notes

1. ../modelnl/machines.html

Chapter 8. Troubleshooting

Troubleshooting create_newcase

Generally, create_newcase errors are reported to the terminal and should provide some guidance about what caused the error.

If create_newcase fails on a relatively generic error, first check carefully that the command line arguments match the interfaces specification. Type

```
> create_newcase -help
```

and review usage.

Troubleshooting job submission problems

This section addresses problems with job submission. Most of the problems associated with submission or launch are very site specific.

First, make sure the runscript, $CASE.$MACH.run, is submitted using the correct batch job submission tool, whether that's qsub, bsub, or something else, and for instance, whether a redirection "<" character is required or not.

Review the batch submission options being used. These probably appear at the top of the $CASE.$MACH.run script but also may be set on the command line when submitting a job. Confirm that the options are consistent with the site specific batch environment, and that the queue names, time limits, and hardware processor request makes sense and is consistent with the case running.

Review the job launch command in the $CASE.$MACH.run script to make sure it's consistent with the site specific recommended tool. This command is usually an mprun, mpiexec, aprun, or something similar. It can be found just after the string "EXECUTION BEGINS HERE" in the $CASE.$MACH.run script.

The batch and run aspects of the $CASE.$MACH.run script is created by the setup script and uses a machine specific mkbatch.$MACH script in the $CCSMROOT/scripts/ccsm_utils/Machines directory. If the run script is not producing correct batch scripts or job launching commands, the mkbatch.$MACH script probably needs to be updated.

Troubleshooting runtime problems

To check that a run completed successfully, check the last several lines of the cpl.log file for the string " SUCCESSFUL TERMINATION OF CPL7-CCSM ". A successful job also usually copies the log files to the directory $CASEROOT/logs.

> **Note:** The first things to check if a job fails are whether the model timed out, whether a disk quota limit was hit, whether a machine went down, or whether a file system became full. If any of those things happened, take appropriate corrective action and resubmit the job.

If it's not clear any of the above caused a case to fail, then there are several places to look for error messages in CESM1.

- Go the $RUNDIR directory. This directory is set in the env_build.xml file. This is the directory where CESM runs. Each component writes its own log file, and there

should be log files there for every component (i.e. of the form cpl.log.yymmdd-hhmmss). Check each component log file for an error message, especially at the end or near the end of each file.
- Check for a standard out and/or standard error file in the $CASEROOT directory. The standard out/err file often captures a significant amount of extra CESM output and it also often contains significant system output when the job terminates. Sometimes, a useful error message can be found well above the bottom of a large standard out/err file. Backtrack from the bottom in search of an error message.
- Go the $RUNDIR directory. Check for core files and review them using an appropriate tool.
- Check any automated email from the job about why a job failed. This is sent by the batch scheduler and is a site specific feature that may or may not exist.
- Check the archive directory. If a case failed, the log files or data may still have been archived. The archiver is turned on if DOUT_S is set to TRUE in env_run.xml. The archive directory is set by the env variable DOUT_S_ROOT and the directory to check is $DOUT_S_ROOT/$CASE.

A common error is for the job to time out which often produces minimal error messages. By reviewing the daily model date stamps in the cpl.log file and the time stamps of files in the $RUNDIR directory, there should be enough information to deduce the start and stop time of a run. If the model was running fine, but the batch wall limit was reached, either reduce the run length or increase the batch time limit request. If the model hangs and then times out, that's usually indicative of either a system problem (an MPI or file system problem) or possibly a model problem. If a system problem is suspected, try to resubmit the job to see if an intermittent problem occurred. Also send help to local site consultants to provide them with feedback about system problems and to get help.

Another error that can cause a timeout is a slow or intermittently slow node. The cpl.log file normally outputs the time used for every model simulation day. To review that data, grep the cpl.log file for the string, tStamp

```
> grep tStamp cpl.log.* | more
```

which gives output that looks like this:

```
tStamp_write: model date = 10120 0 wall clock = 2009-09-28 09:10:46 avg dt = 58.58 dt =
tStamp_write: model date = 10121 0 wall clock = 2009-09-28 09:12:32 avg dt = 60.10 dt =
```

and review the run times for each model day. These are indicated at the end of each line. The "avg dt = " is the running average time to simulate a model day in the current run and "dt = " is the time needed to simulate the latest model day. The model date is printed in YYYYMMDD format and the wall clock is the local date and time. So in this case 10120 is Jan 20, 0001, and it took 58 seconds to run that day. The next day, Jan 21, took 105.90 seconds. If a wide variation in the simulation time is observed for typical mid-month model days, then that is suggestive of a system problem. However, be aware that there are variations in the cost of the CESM1 model over time. For instance, on the last day of every simulated month, CESM1 typically write netcdf files, and this can be a significant intermittent cost. Also, some models read data mid month or run physics intermittently at a timestep longer than one day. In those cases, some run time variability would be observed and it would be caused by CESM1, not system variability. With system performance variability, the time variation is typically quite erratic and unpredictable.

Sometimes when a job times out, or it overflows disk space, the restart files will get mangled. With the exception of the CAM and CLM history files, all the restart files have consistent sizes. Compare the restart files against the sizes of a previous restart. If they don't match, then remove them and move the previous restart into place before resubmitting the job. Please see restarting a run.

On HPC systems, it is not completely uncommon for nodes to fail or for access to large file systems to hang. Please make sure a case fails consistently in the same place

before filing a bug report with CESM1.

Additional Troubleshooting Information

The Community Land Model (CLM) documentation, Chapter 6.[1], provides some additional trouble shooting tips that may be useful beyond those listed here.

Notes

1. http://www.cesm.ucar.edu/models/cesm1.2/clm/models/lnd/clm/doc/UsersGuide/c12493.html

Glossary

Branch

One of the three ways to initialize CESM runs. CESM runs can be started as startup, hybrid or branch. Branch runs use the BRANCH $RUN_TYPE and the restart files from a previous run. See **setting up a branch run.**

case

The name of your job. This can be any string.

$CASE

The case name. But when running **create_newcase**, it doubles as the case directory path name where build and run scripts are placed. $CASE is defined when you execute the **create_newcase** command;, and set in `env_case.xml`. Please see create a new case.

$CASEROOT

$CASEROOT - the full pathname of the root directory for case scripts (e.g. /user/$CASE). You define $CASEROOT when you execute the **create_newcase**, and is set in `env_case.xml`. $CASEROOT must be unique.

component

component - Each model can be run with one of several components. Examples of components include CAM, CICE, CLM, and POP. Component names will always be in all caps.

component set (compset)

Preset configuration settings for a model run. These are defined in component sets[1].

$CCSMROOT

The full pathname of the root directory of the CESM source code. $CCSMROOT is defined when you execute the **create_newcase** command.

$EXEROOT

The case executable directory root. $EXEROOT is defined when you execute the **configure** command, and is set in `env_build.xml`.

hybrid run

A type of run. Hybrid runs use the HYBRID $RUN_TYPE and are initialized as an initial run, but use restart datasets from a previous CESM case. Please see **setting up a hybrid run.**

$MACH

The supported machine name, and is defined in `env_case.xml` when you run the configure command. Please see hardware platforms[2] for the list of supported machines.

model

CESM is comprised of five models (atm, ice, glc, lnd, ocn) and the coupler, cpl. The word model is loosely used to mean any one of the models or the coupler.

model input data

Refers to static input data for the model. Input data are provided as part of the release via an inputdata area or data from a server.

release

A supported version of CESM.

restart

Refers to a set of data files and pointer files that the model generates and uses to restart a case.

$RUNDIR

The directory where the model is run, output data files are created, and log files are found when the job fails. This is usually $EXEROOT/run, and is set in env_build.xml by the configure command.

tag

A version of CESM. Note: A tag may not be supported.

Notes

1. ../cesm1.1/cesm/modelnl/compsets.html
2. ../modelnl/machines.html