CCSM3 Scripts Tutorial
How to Build, Run, and Test
CCSM3

http://www.cccsm.ucar.edu/models/ccsm3.0/ccsm/

CCSM Software Engineering Group
Climate and Global Dynamics Division
NCAR
Tutorial Outline

- User Support and Post Processing
  Sylvia Murphy
- How to Build and Run CCSM3
  Mariana Vertenstein
- How to Test CCSM3
  Tom Henderson
- Setting up Production Runs
  Lawrence Buja
- Machine Dependent Details
  George R Carr Jr
User Support

- Send user support questions to ccsm@ucar.edu
- User’s Guide
  http://www.ccsm.ucar.edu/models/ccsm3.0
- Post Processing:
  - netCDF Operators (NCO)
  - NCAR Command Language (NCL)
    - http://www.ncl.ucar.edu
How to Build and Run

CCSM3

Mariana Vertenstein
Climate and Global Dynamics Division
NCAR
mvertens@ucar.edu
Building and Running CCSM3

I. What is CCSM?
II. How do I get started?
III. CCSM3 Script Features
IV. How do I build and run a case?
V. More details …
CCSM Models

atm

Ind

cpl

ocn

ice
CCSM Components

- each model is represented by components:
  - atm: cam, datm, latm, xatm
  - lnd: clm, dlnnd, xlnnd
  - ice: csim, dice, xice
  - ocn: pop, docn, xocn
  - cpl: cpl
- active components are cam, clm, csim, pop
- "dead" components (xatm..) are used for software testing
Building and Running CCSM3

I. What is CCSM?

II. How do I get started?

III. CCSM3 Script Features

IV. How do I build and run a case?

V. More details ...
The CCSM3.0 distribution

Model code and scripts:

ccsm3.0.tar.gz

Input data (e.g. for T42_gx1v3):

ccsm3.0.inputdata.atm_lnd.tar.gz
ccsm3.0.inputdata.T42.tar.gz
ccsm3.0.inputdata.gx1v3.tar.gz
ccsm3.0.inputdata.cpl.tar.gz

http://www.ccsm.ucar.edu/models/ccsm3.0
CCSM3 Case Directories

- Long-term archiving
  - $DOUT_MSROOT/

- Short-term archiving
  - $DOUT_S_ROOT/

- Source code
  - $CCSMROOT/

- Case Scripts
  - $CASEROOT/

- Run Directory
  - $EXEROOT/

- Input data
  - $DIN_LOC_ROOT/

7/22/04
CCSM3.0 top level script directory

$HOME/ccsm3 ($CCSMROOT/)

scripts/

create_newcase  create_test  README

ccsm_utils/
CCSM3.0 input data directory

$(HOME)/inputdata/ ($DIN_LOC_ROOT)

cpl/
cpl6/
atm/
cam2/
datm6/
latm6/

Above directories have not kept up with component names
Building and Running CCSM3

I. What is CCSM?
II. How do I get started?
III. CCSM3 Script Features
IV. How do I build and run a case?
V. More details …
Key script features (1 of 2)

- modular and extensible - easy to use
- modular - cases are generated using 3 editable environment variable files
  - build time related environment file
  - run time environment file
  - machine dependent environment file
    - (contains both build and run time variables)
- extensible - straightforward to add new machines
- script error checking adds reliability
Key script features (2 of 2)

- Default MPI tasks/ OpenMP threads are provided for each component, resolution and machine
- User can run same case on multiple machines out of one case directory
- Larger set of supported resolutions
- Automated tests are provided
- Performance tools are included
Building and Running CCSM3

I. What is CCSM?
II. How do I get started?
III. CCSM3 Script Features
IV. How do I build and run a case?
V. More details ...
Naming Conventions

- $CCSMROOT/ - root directory containing CCSM3 source code and scripts
- $CASE - defines both new case name AND case directory name
- $CASEROOT/ - root of new case directory (e.g. $HOME/$CASE)
- $MACH - "supported" machine name
- $EXEROOT/ - root directory containing model executables
Script Basics

- Two commands generate new case scripts
- `create_newcase`
  - creates a new CCSM3 case directory containing 3 environment variable files
- `configure`
  - uses the environment variable files to generate build and run scripts
Creating Building and Running a CCSM3 case

Step 1: cd into the scripts directory and create a new case for the target machine

Step 2: cd into the new case directory and configure the new case for the target machine

Step 3: build the model on the target machine

Step 4: run the model on the target machine

Step 5: examine output data
Default case - 6 commands

To generate a T42_gx1v3 1990 control run with fully active components that will run for 5 days on NCAR IBM-SP blackforest

> cd $CCSMROOT/scripts
> ./create_newcase -case /user/Case1 -mach blackforest
> cd /user/Case1
> ./configure -mach blackforest
> ./Case1.blackforest.build
> llsubmit Case1.blackforest.run
Step1: create_newcase produces:

- new /user/Case1 directory containing:
  - configure
  - env_conf
  - env_run
  - env_mach.blackforest
  - env.readme
  - SourceMods/

- $CASEROOT is /user/Case1
- $CASE is Case1
- $MACH is blackforest
- SourceMods/ - place holder for user-modified source code
Step2: configure produces

$CASEROOT/

$CASE.$MACH.build
- Buildnml_prestage/
  - cam.buildnml_prestage.csh
  - clm.buildnml_prestage.csh
  - cpl.buildnml_prestage.csh
  - csim.buildnml_prestage.csh
  - pop.buildnml_prestage.csh

$CASE.$MACH.run
- Buildexe/
  - cam.buildexe.csh
  - clm.buildexe.csh
  - cpl.buildexe.csh
  - csim.buildexe.csh
  - pop.buildexe.csh

$CASE.$MACH.l_archive
- Buildlib/
  - esmf.buildlib
  - mct.buildlib
  - mph.buildlib
Step3: Build the CCSM3 model

- prestages input data in $EXEROOT/
- creates component namelist in $EXEROOT/
- creates component executables in $EXEROOT/
  - $EXEROOT/atm/<atm exec>
  - $EXEROOT/Ind/<land exec>
  - $EXEROOT/ocn/<ocn exec>
  - $EXEROOT/ice/<ice exec>
  - $EXEROOT/cpl/cpl
Step 4 - Run the CCSM3 model

- submit $CASE.$MACH.run to batch queue
  - `cd /user/Case1`
  - `llsubmit Case1.blackforest.run`
- invoke build script and submit run script from $CASEROOT/
- model will be run in $EXEROOT/
Building and Running CCSM3

I. What is CCSM?
II. How do I get started?
III. CCSM3 Script Features
IV. How do I build and run a case?
V. More details ...
CCSM3 Component Parallelization

- **CAM**: MPI, OpenMP or MPI/OpenMP
- **CLM**: MPI, OpenMP or MPI/OpenMP
- **CSIM**: MPI only
- **POP**: MPI only
- **CPL**: MPI, OpenMP or MPI/OpenMP
- **Data and Dead Comps**: serial (1 proc)
CCSM3 Component Resolutions

- **cam/clm**: T85, T42, T31, 2x2.5
- **datm/dlnd**: T42, T31
- **latm/dlnd**: T62
- **pop/csim**: gx1v3, gx3v5
- **docn/dice**: gx1v3, gx3v5
CCSM3 Model Resolutions

Component resolutions can be combined as follows:

- T85_gx1v3
- T42_gx1v3, T42_gx3v5
- T31_gx3v5
- 2x2.5_gx1v3 (cam finite volume only)
- T62_gx1v3, T62_gx3v5 (latm only)
CCSM3 Component Sets

- A = datm, dInd, docn, dice, cpl
- B = cam, clm, pop, csim, cpl
- C = datm, dInd, pop, dice, cpl
- D = datm, dInd, docn, csim, cpl
- G = latm, dInd, pop, csim, cpl
- H = cam, dInd, docn, dice, cpl
- I = datm, clm, docn, dice, cpl
- K = cam, clm, docn, dice, cpl
- L = latm, dInd, pop, dice, cpl
- M = latm, dInd, docn, csim (mixed layer ocean), cpl
- O = latm, dInd, docn, dice, cpl
- X = xatm, xInd, xocn, xice, cpl
Details of Building and Running a case

Step 1: cd into $CCSMROOT/scripts/ and run create_newcase

Step 2: cd into $CASEROOT/, optionally edit env_conf and tasks/threads in env_mach.$MACH, run configure

Step 3: build CCSM3 model interactively by running $CASE.$MACH.build

Step 4: optionally edit env_run and non-task/thread part of env_mach.$MACH, submit $CASE.$MACH.run

Step 5: examine output data
Step 1: run create_newcase

> cd $CCSMROOT/scripts/
> create_newcase -case $CASEROOT -mach $MACH [-compset <comp set>] [-res <resolution>]

$CCSMROOT and $CASEROOT => env_run
$MACH => env_mach.$MACH
resolution and comp set => env_conf
Step 2: configure command

- **configure** creates build and run scripts using environment variables in
  - `env_conf` and `env_mach.$MACH`

- **edit before running configure:**
  - `env_conf`
  - MPI tasks/OpenMP threads in `env_mach.$MACH`

- **edit anytime:**
  - `env_run`
  - non tasks/threads in `env_mach.$MACH`
## Step 2: Edit env_conf (1 of 2)

<table>
<thead>
<tr>
<th>Environment var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CASE</td>
<td>case name</td>
</tr>
<tr>
<td>$CASESTR</td>
<td>case description</td>
</tr>
<tr>
<td>$COMP_ATM</td>
<td>atm comp: cam, datm, latm, xatm</td>
</tr>
<tr>
<td>$COMP_LND</td>
<td>lnd component: clm, dlnd, xlnd</td>
</tr>
<tr>
<td>$COMP_ICE</td>
<td>ice component: csim, dice, xice</td>
</tr>
<tr>
<td>$COMP_OCN</td>
<td>ocn component: pop, docn, xocn</td>
</tr>
<tr>
<td>$COMP_CPL</td>
<td>cpl component: cpl</td>
</tr>
<tr>
<td>$CSIM_MODE</td>
<td>Prognostic, oceanmixed_ice</td>
</tr>
</tbody>
</table>
## Step 2: Edit env_conf (2 of 2)

<table>
<thead>
<tr>
<th>Environment Var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RUN_TYPE</td>
<td>startup, branch, hybrid</td>
</tr>
<tr>
<td>$RUN_STARTDATE</td>
<td>yyyy-mm-dd (startup or hybrid)</td>
</tr>
<tr>
<td>$RUN_REFCASE</td>
<td>Ref case name (branch or hybrid)</td>
</tr>
<tr>
<td>$RUN_REFDATE</td>
<td>Ref yyyy-mm-dd (branch or hybrid)</td>
</tr>
<tr>
<td>$GRID</td>
<td>T42_gx1v3, T85_gx1v3, ...</td>
</tr>
<tr>
<td>$IPCC_MODE</td>
<td>OFF, 1870_CONTROL, RAMP_CO2_ONLY</td>
</tr>
</tbody>
</table>
Step 2: Edit env_mach.$MACH

tasks/threads

Machine specific settings provided for MPI/tasks and OpenMP threads. If default settings are changed, modify:

- setenv NTASKS_ATM $ntasks_atm
- setenv NTHRDS_ATM $nthrds_atm
- setenv NTASKS_LND $ntasks_lnd
- setenv NTHRDS_LND $nthrds_lnd
- setenv NTASKS_OCN $ntasks_ocn
- setenv NTHRDS_OCN $nthrds_ocn
- setenv NTASKS_ICE $ntasks_ice
- setenv NTHRDS_ICE $nthrds_ice
Step 2: Resolved scripts (1 of 2)

- The `configure` command generates “resolved” scripts in `Buildnml_prestage/` and `Buildexe/` valid for given component set, resolution, CCSM initialization (set by `env_conf` variables).

- If want to change `env_conf` after running `configure` - must use:
  - `configure -cleanall`
  - `configure -mach $MACH`
configure also uses tasks/threads in env_mach.$MACH to produce batch queue command - on ibm

# @ task_geometry = {((......))}

if change tasks/threads in env_mach.$MACH after running configure must use

> configure -cleanmach $MACH
> configure -mach $MACH
Step 2: CCSM Initialization

- Initialization set by $RUN_TYPE$ in env_conf
  - startup: new run from “cold start” input files
  - hybrid: new run from combination of initial (cam, clm) and restart files (pop, csim)
  - branch: new run from restart files

- Each initialization type has a unique set of input data

- Continuation run set by $CONTINUE_RUN$ in env_run
Step 3: Build the CCSM model

- Build the CCSM3 model interactively
  
  > ./$CASE.$MACH.build

- Calls Buildlib/*buildlib

- Calls Buildnml_prestage/*.buildnml_prestage.csh
  
  - Prestages necessary input data
  - Copies data $DIN_LOC_ROOT -> $EXEROOT
  - $DIN_LOC_ROOT needs to be accessible from $EXEROOT

- Calls Buildexe/*.buildexe.csh
  
  - each *.buildexe.csh calls gmake
Step 3: Build Macros and Makefile

- All component executables created using Makefile in `$CCSMROOT/models/bld/
- Makefile is machine independent - uses machine specific details in Macros.$OS files (e.g., Macros.AIX)
- User should modify appropriate Macros.$OS file to change machine specific gmake flags
Step 4: Run the CCSM model

- Submit (or run) $CASE.$MACH.run
- $CASE.$MACH.build is invoked from $CASE.$MACH.run
- Input data will be prestaged from $DIN_LOC_ROOT/ during build
- Model will be run in $EXEROOT/
- Component stop time and restart file write times controlled by $STOP_OPTION and $STOP_N
Step 4: Short and Long-term archiving

- Short-term archiving moves model output data to separate area on local disk
  - set by $DOUT_S_ROOT$

- Long-term archiving copies model output data from $DOUT_S_ROOT$ to local mass store
  - set by $DOUT_L_MSROOT$
  - done by script $CASE.$MACH.l_archive
# Step 4: Edit env_run

<table>
<thead>
<tr>
<th>Environment Var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RESUBMIT</td>
<td>Automatic resubmission number</td>
</tr>
<tr>
<td>$CONTINUE_RUN</td>
<td>Continuation run flag</td>
</tr>
<tr>
<td></td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>$STOP_OPTION</td>
<td>Coupler stop time (ndays, nmonths, newyear...)</td>
</tr>
<tr>
<td>$STOP_N</td>
<td>Number of days or months</td>
</tr>
</tbody>
</table>
### Step 4: Edit env_mach.$mach

<table>
<thead>
<tr>
<th>Environment var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$EXEROOT</code></td>
<td>Executable root dir</td>
</tr>
<tr>
<td><code>$DIN_LOC_ROOT</code></td>
<td>Input data root dir</td>
</tr>
<tr>
<td><code>$DOUT_S</code></td>
<td>Turns on short-term archiving</td>
</tr>
<tr>
<td><code>$DOUT_S_ROOT</code></td>
<td>Short-term archiving root</td>
</tr>
<tr>
<td><code>$DOUT_L_MS</code></td>
<td>Turns on long-term archiving</td>
</tr>
<tr>
<td><code>$DOUT_L_MS_ROOT</code></td>
<td>Long-term archiving root</td>
</tr>
</tbody>
</table>
Step 5: Model output data

- only active components output history and restart files (default)
- active components write netCDF monthly averaged history files (default)
- active components write binary restart files at end of run (default)
- CAM and CLM also periodically write netCDF initial files at beginning of each year (default)
- each component writes standard output “log” files
Summary

CCSM is now easier to build and run!

For more details see the CCSM3.0 User’s Guide at:
www.ccsm.ucar.edu/models/ccsm3.0
CCSM3 Testing

Tom Henderson
Climate and Global Dynamics Division
NCAR
hender@ucar.edu
Overview

- Built-in test facility
- CCSM3 test cases
- Test creation
- Test execution
- Test evaluation
Built-in Test Facility (1 of 2)

- CCSM includes new built-in tests
- Everyone should use them
- Why should users test?
  - Validate download
    - Source code, data sets, etc.
  - Verify exact restart after all source code changes
- See User’s Guide section 7
Built-in Test Facility (2 of 2)

- DO NOT USE BUILT-IN-TEST SCRIPTS TO START PRODUCTION RUNS
- Only use `create_newcase` for production runs
Common CCSM Test Cases (1 of 2)

- Smoke test
  - Run for a few days
  - Pass if run completes

- Exact restart
  - Compare initial and restart runs
  - Pass if results match bit-for-bit
Common CCSM Test Cases (2 of 2)

- **Debug**
  - Run with compiler trapping
    - Out-of-bounds indexing, floating-point exceptions, etc.
  - Pass if run completes

- **Regression**
  - Compare with old run
  - Pass if results match bit-for-bit
Test Case Names (1 of 2)

- Exact restart for startup runs
  - ER.01a: 1990 control
  - ER.01b: 1870 control
  - ER.01e: CO2 ramping

- Exact restart for branch runs
  - BR.02a: 1990 control

- Exact restart for hybrid runs
  - HY.02a: 1990 control
Test Case Names (2 of 2)

- Debug (software trapping)
  - DB.01a: 1990 control
  - DB.01b: 1870 control
  - DB.01e: CO2 ramping
Test Creation (1 of 3)

Choose a test case (like ER.01a), then select:
- Resolution
  - T31_gx3v5, T42_gx1v3, T85_gx1v3, ...
- Machine
  - bluesky, blackforest, chinook, jazz, ...
- Component set
  - B, A, X, G, ...
Test Creation (2 of 3)

- Run `create_test` from CCSM3 scripts directory
  - Uses `create_newcase` and `configure`
- Try the `-help` option...
- Use the `-testroot` option to specify location of generated test scripts
  - Otherwise location is in CCSM3 scripts directory
Test Creation (3 of 3)

- Use the **-inputdataroot** option to specify alternate input data directory
  - Use outside of NCAR
Test Creation Example (1 of 3)

- Exact restart test for 1990 control

> create_test -test ER.01a -mach bluesky
  -res T42_gx1v3 -compset B
  -testroot $HOME/tst

... Successfully created new case root directory
  $HOME/tst/TER.01a.T42_gx1v3.B.bluesky.123456
...
Test Creation Example (2 of 3)

- create_test...
  - Creates new test directory
  - Runs create_newcase and configure
    - Creates usual build and run scripts
    - Do not use run script!
  - Builds test script
  - Also builds script batch.$MACH
    - Use to run test suite, see Users Guide
Test Creation Example (3 of 3)

> cd $HOME/tst/TER.01a.T42_gx1v3.B.bluesky.123456
> ls
TER.01a.T42_gx1v3.bluesky.B.123456.build
TER.01a.T42_gx1v3.bluesky.B.123456.run
TER.01a.T42_gx1v3.bluesky.B.123456.test
configure
restart_compare.pl
env_run
Buildsnml_Prestage/
Buildexe/
...
Test Execution Example (1 of 2)

- Go to new test case directory

  > cd $HOME/tst/TER.01a.T42_gx1v3.B.bluesky.123456

- Run build script interactively

  > TER.01a.T42_gx1v3.bluesky.B.123456.build
Test Execution Example (2 of 2)

- Edit test script to modify default batch queue (optional)

  > vi TER.01a.T42_gx1v3.B.bluesky.123456.test

- Submit test script to queue

  > llsubmit TER.01a.T42_gx1v3.B.bluesky.123456.test
Test Evaluation

- Test results summarized in two files
  - Testcase.out
    - Human-readable log file
  - Testcase
    - Simple state
      - PASS, FAIL, ERROR, ...
    - “PASS” means test passed
    - Anything else means look at log file
Test Evaluation Example

> more Testcase
PASS

> more Testcase.out
doing a 10 day initial test
...
Doing a 5 day restart test
...
Comparing initial log file with
  restart/branch/hybrid log file...
log files match!
PASS
Questions?

- See section 7 in the User’s Guide
CCSM3 Production Runs

Lawrence Buja
Climate Change Working Group
southern@ucar.edu
CCSM3 Case Directories

Long-term archiving
$DOUT_MSROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Run Directory
$EXEROOT/

Short-term archiving
$DOUT_S_ROOT/

CCSM CASE

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

CCSM CASE

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/

Long-term archiving
$DOUT_MSROOT/

Run Directory
$EXEROOT/

Source code (frozen!)
$CCSMROOT/

Case Scripts
$CASEROOT/

Input data
$DIN_LOC_ROOT/
Resources

- **Computational Cost**: (on NCAR Bluesky, 1 GAU = $21):

<table>
<thead>
<tr>
<th>Resolution</th>
<th>PEs</th>
<th>Myr/day</th>
<th>CPUhrs/Myr</th>
<th>GAUs/Myr</th>
</tr>
</thead>
<tbody>
<tr>
<td>T31_gx3v5</td>
<td>64</td>
<td>24.2</td>
<td>63</td>
<td>16</td>
</tr>
<tr>
<td>T42_gx1v3</td>
<td>128</td>
<td>8.5</td>
<td>363</td>
<td>87</td>
</tr>
<tr>
<td>T85_gx1v3</td>
<td>128</td>
<td>3.0</td>
<td>1030</td>
<td>244</td>
</tr>
</tbody>
</table>

- **Data Volume**:  
  - T42_gx1v3 = 6 GB/Myr  
  - T85_gx1v3 = 10 GB/Myr

- **Disk space**:  
  - 100 GB of scratch space per case

- **Time to Solution** = \( Y / S + Q + D \)
  
  - \( Y \) = Years of integration
  - \( S \) = Ave Model Execution Rate ( Model_Year/Wall_Day )
  - \( Q \) = Queue wait time
  - \( D \) = Machine downtime
CCSM Data Flow

INPUT:
- CCSM run scripts ($CASEROOT/$CASE.$MACH.run)
- Initial/Restart data (usually previous CCSM run)
- Boundary Data (Located in $DIN_LOC_ROOT)

Output Data Archiving ($DOUT_S_ROOT)
- History data
- Restart data
- Initial data
- Printed Log files
CCSM T85 Data Output

T85 IPCC: 9.6 Gbytes/year

Submit CCSM run script
$CASEROOT/$CASE.$MACH.run

Super

cpl

0.1 Gbytes

4.5 Gbytes

atm

4.2 Gbytes

ocn

0.5 Gbytes

ice

0.3 Gbytes

Tape Archive Devices

$EXEROOT/

$DOUT_S_ROOT/

$DOUT_MSROOT/

Front end

$CASEROOT/$CASE.$MACH.run

9.6 GB/yr
Setting up a run

1. Standard Build + some modification
   - `$CCSMROOT/scripts/create_newcase`
   - Modify `env_conf`, `env_run`, `env_mach.$MACH`
   - configure
   - Apply modification to scripts or code

2. Prestage initial/restart datasets

3. Build model interactively
   (run `$CASEROOT/$CASE.$MACH.build`)

4. Check that modifications happen
   - Look in `$EXEROOT/*/.*.buildexe.*`

5. Check exact restartability
Setting Up a Production Run

1. `cd $CCSMROOT/scripts`
2. `.create_newcase -case $CASEROOT -mach $MACH -res T85_gx1v3 -compset B`
3. `cd $CASEROOT`
4. `Modify env_conf, env_run, env_mach.$MACH`
5. `.configure -cleanall`
6. `.configure -mach $MACH`
7. `Modify $CASEROOT/Buildnml_Prestage/*.csh as necessary`
8. `Position restart files in $DOUT_S_ROOT/restart`
9. `Build model interactively: $CASEROOT/$CASE.$MACH.build`
10. `Submit $CASEROOT/$CASE.$MACH.run`
### Production env_conf settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_TYPE</td>
<td>branch</td>
</tr>
<tr>
<td>RUN_STARTDATE</td>
<td>2000-01-01</td>
</tr>
<tr>
<td>RUN_REFCASE</td>
<td>b30.030a</td>
</tr>
<tr>
<td>RUN_REFDATE</td>
<td>2000-01-01</td>
</tr>
<tr>
<td>CASESTR</td>
<td>&quot;$GRID $IPCC_MODE from $RUN_REFCASE year $RUN_REFDATE&quot;</td>
</tr>
</tbody>
</table>
Production env_run settings

CASEROOT          $HOME/ccsm_runs/$CASE
CCSMROOT          $HOME/ccsm3
STOP_OPTION       yearly
INFO_DBG          0
DIAG_N            365
### Production env_mach settings

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEROOT</td>
<td>$SCRATCH/$LOGNAME/$CASE</td>
</tr>
<tr>
<td>DIN_LOC_ROOT</td>
<td>$HOME/ccsm/inputdata</td>
</tr>
<tr>
<td>DOUT_S</td>
<td>TRUE</td>
</tr>
<tr>
<td>DOUT_L_MS</td>
<td>TRUE</td>
</tr>
<tr>
<td>DOUT_L_MSNAME</td>
<td>CCSM</td>
</tr>
<tr>
<td>DOUT_L_MSPWD</td>
<td>secret</td>
</tr>
</tbody>
</table>
Modifying CCSM

- Changing code
  - Frozen code (Don’t modify code in $CCSMROOT!)
  - Modified code ($CASEROOT/Source/Mods/src.* )
- Changing boundary data
  - Modify
    $CASEROOT/Buildnml_Prestage/cam.buildnml_prestage.csh
- Validating your change
  - Verify that your change was applied correctly
  - Document your change
  - Do no damage:
    - Check exact restarts
    - Check performance
    - Compare new model climate with control climate
Running a production job

- Run in batch queues
  ```
  llsubmit $CASEROOT/$CASE.$MACH.run
  ```

- Extend CCSM runs as "continue" runs.
  - A continued run gives exactly the same results as if the run had never stopped.
  - Set CONTINUE_RUN TRUE in $CASEROOT/env_run

- CCSM Restart Data
  - CCSM writes restart files at specified intervals
  - $DOUT_S_ROOT/restart/ & restart.tars/
Automatic resubmission

- The CCSM can automatically resubmit itself

- RESUBMIT variable:
  - Automatic resubmit flag that counts down to 0
  - Located in $CASEROOT/env_run

- At end of run, if RESUBMIT is not 0, automatically:
  - Decrement RESUBMIT by 1 ($CASEROOT/env_run)
  - Set CONTINUE_RUN true ($CASEROOT/env_run)
  - Resubmit $CASEROOT/$CASE.$MACH.run
Monitoring a production job

- Is it still running?
  - Check your job(s) in the batch queue:
    `llq -u $LOGNAME` (IBM SP)
  - Monitor the end of the newest cpl log file
    `tail -30 `ls -t $EXEROOT/cpl/cpl.log.* | head -1``

- Disk space management
  - Verify long-term archiving
  - Monitor quotas: `spquota (bluesky)`
  - Monitor disk usage: `du $EXEDIR/.. | sort -n`
  - Cleanout `$LOGDIR`
Monitoring a production job

b30.036a Annual Sea-Ice Area [Jun 28 2004]

SH Std Dev = 0.592

NH Std Dev = 0.834


Model Year

Area x 10^6 km²
9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0

- HadISST NH
- Model NH
- HadISST SH
- Model SH
Log Files, Aborts and Errors

- **Common Model Errors:**
  - Build failure
  - Pointing to wrong restart directories
  - No/incorrect input data
  - POP ocean model non-convergence
  - CAM model stops due to non-convergence
  - CSIM model failures

- **Exceed disk quotas or Wall-clock limits**

- **System problems**

- **Warnings:**
  - CAM Courant limit warning messages
Log Files, Aborts and Errors

Finding your Error can be a challenge!

Look at:

1. $EXEROOT/*/*.*.log.*
2. $CASEROOT/poe.std*
3. Your mailbox
4. quotas, batch queue limits, disk scrubbing

Some of my favorite shortcuts:

- alias mev 'source env_run;source env_run;source env_mach.bluesky32'
- alias s 'cd $CASEROOT; ls -lrt | tail -20'
- alias e 'cd $EXEROOT; ls -ldrt /*/* | tail -20'
- alias r 'cd $DOUT_S_ROOT/restart; ls -lrt'
- alias tc 'tail -30 `ls -rt */cpl.log.* | tail -1`'
- alias mo 'more `ls -rt $CASEROOT/poe.stdout.* | tail -1`'
- alias me 'more +/" C O N N" `ls -rt $CASEROOT/poe.stderr.* | tail -1`
Questions?

- See use cases in the User’s Guide
Machine Dependent Details

George R Carr Jr
Climate and Global Dynamics Division
NCAR
gcarr@ucar.edu
Where Do You Start???

- You’ve got a machine to run on
- You’ve got users (might be you)
- You’ve got the tarball’s (src and data)
- You’ve read the User Guide ... right?
- You want to make building and running CCSM3 on your system .... EASY
The Process

- Look at the list of machines already supported
- You may choose to ...
  - Make your machine look like a fully supported machine
  - Create/modify machine specific files based on the existing files for other similar machine(s)
    - You’ll need to know some of the basic details of your system (libraries, tools, architecture, ... )
Categories of Machine Support (1 of 2)

1. Climate Verified, fully tested
   - bluesky, bluesky32, blackforest, cheetah, seaborg

2. Runs, passes exact restart test, climate not verified
   - chinook, jazz
Machines Supported (2 of 2)

3 Builds, might not run, may not pass exact restart test
   - anchorage, bangkok, phoenix, lemieux, moon

4 Unsupported, being looked at or worked on, possible future support
   - eagle, ram, calgary, mauve, rime, lodestone, rex, TBD Apple G5, many others
## Machine Descriptions

<table>
<thead>
<tr>
<th>Machine</th>
<th>Description</th>
<th>OS</th>
<th>Compiler</th>
<th>Network type</th>
<th>Queue SW</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>bluesky</td>
<td>IBM Power4</td>
<td>AIX</td>
<td>IBM XL</td>
<td>IBM</td>
<td>Load Leveler</td>
<td>1</td>
</tr>
<tr>
<td>bluesky32</td>
<td>IBM Power4</td>
<td>AIX</td>
<td>IBM XL</td>
<td>IBM</td>
<td>Load Leveler</td>
<td>1</td>
</tr>
<tr>
<td>blackforest</td>
<td>IBM Power3</td>
<td>AIX</td>
<td>IBM XL</td>
<td>IBM</td>
<td>Load Leveler</td>
<td>1</td>
</tr>
<tr>
<td>cheetah</td>
<td>IBM Power4</td>
<td>AIX</td>
<td>IBM XL</td>
<td>IBM</td>
<td>Load Leveler</td>
<td>1</td>
</tr>
<tr>
<td>seaborg</td>
<td>IBM Power3</td>
<td>AIX</td>
<td>IBM XL</td>
<td>IBM</td>
<td>Load Leveler</td>
<td>1</td>
</tr>
<tr>
<td>chinook</td>
<td>SGI R12000</td>
<td>IRIX</td>
<td>MIPS</td>
<td>NumaLink</td>
<td>PBS</td>
<td>2</td>
</tr>
<tr>
<td>jazz</td>
<td>Intel Xeon</td>
<td>Linux</td>
<td>PGI</td>
<td>Myrinet</td>
<td>PBS</td>
<td>2</td>
</tr>
<tr>
<td>anchorage</td>
<td>Intel Xeon</td>
<td>Linux</td>
<td>PGI</td>
<td>Gbit Ethernet</td>
<td>SPBS</td>
<td>3</td>
</tr>
<tr>
<td>bangkok</td>
<td>Intel Xeon</td>
<td>Linux</td>
<td>PGI</td>
<td>Gbit Ethernet</td>
<td>SPBS</td>
<td>3</td>
</tr>
<tr>
<td>lemeieux</td>
<td>Alpha</td>
<td>OSF/1</td>
<td>Compaq</td>
<td>Myrinet</td>
<td>PBS</td>
<td>3</td>
</tr>
<tr>
<td>moon</td>
<td>Earth Simulator</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
<td>PBS</td>
<td>3</td>
</tr>
<tr>
<td>phoenix</td>
<td>Cray X1</td>
<td>Unicos</td>
<td>Cray</td>
<td>CrayLink</td>
<td>PBS</td>
<td>4</td>
</tr>
<tr>
<td>calgary</td>
<td>Intel Xeon</td>
<td>Linux</td>
<td>PGI</td>
<td>InfiniBand</td>
<td>SPBS</td>
<td>4</td>
</tr>
<tr>
<td>rex</td>
<td>AMD Opteron</td>
<td>Linux</td>
<td>PGI</td>
<td>Myrinet</td>
<td>PBS</td>
<td>4</td>
</tr>
<tr>
<td>lightning</td>
<td>AMD Opteron</td>
<td>Linux</td>
<td>PGI</td>
<td>Myrinet</td>
<td>Load Sharing Facility</td>
<td>4</td>
</tr>
</tbody>
</table>
What To Expect???

- If your machine matches a supported machine ... order hours to build
- If your machine is similar to a supported machine ... order days to build
- If your machine is completely different ... order days to weeks
Setting Up a New Machine

- Look for supported machine(s) with
  - Your CPU
  - Your architecture
  - Your compiler
  - Your interconnect
  - Your batch process
  - Your storage strategy

- May need to combine from more than one “supported” configuration
Files of Interest

Can be found in

- `$CCSMROOT/scripts/ccsm_utils/Tools/`
- `$CCSMROOT/scripts/ccsm_utils/Machines/`
- `$CCSMROOT/models/bld/`
$HOME/ccsm3  ($CCSMROOT/)
Recipe For A New Machine

Assuming new “Linux” machine named “newmach”

- Modify 1 file
  - `$CCSMROOT/scripts/ccsm_utils/Tools/check_machine` to add “newmach”

- Create 3 files minimum (at most 5 files)
  - `$CCSMROOT/scripts/ccsm_utils/Machines/{batch, run, env, l_archive, modules}.linux.newmach`

- Look at, maybe modify one more file
  - `$CCSMROOT/models/bld/Macros.Linux`

- Try it ... run some of the test scripts

- Repeat as needed
The File “check_machine”

```bash
#!/bin/csh -f

#-------------- valid machine list
set resok = ( \
  bluesky \
  bluesky32 \
  blackforest \
  cheetah \
  cheetah32 \
  eagle \
  ;
  ;
  rime \
  generic_ibm \
  generic_sgi \
  generic_linux \
  generic_compaq \
  generic_sx )

#-----------------------
```

newmach \n
“Machines” Directory Files

What do the file names look like
- \{batch, env, run, l_archive, modules\}.<machine vendor or type>.<machine name>

Examples
- batch.linux.bangkok
- env.linux.lodestone
- run.linux.jazz
- l_archive.ibm.bluesky
- modules.sgi.chinook
“Machines” Directory Files

- batch.* (required) - provides the template for building the batch job submission commands
- env.* (required) - provides the template for basic component configuration and model run options
- run.* (required) - provides the template for building the commands needed to run the model
- l_archive.* (optional) - commands for long term archiving
- modules.* (optional) - specific commands for machines needing run modules
#! /bin/csh -f

# Documentation of following environment variables is provided in env.readme
#
# Tasks and Threads: Edit any time prior to invoking the configure command
#
set COMPONENTS = ($COMP_CPL $COMP_ICE $COMP_LND $COMP_OCN $COMP_ATM)

    set ntasks_atm = 1; set nthrds_atm = 1
    set ntasks_lnd = 1; set nthrds_lnd = 1
    set ntasks_ice = 1; set nthrds_ice = 1
    set ntasks_ocn = 1; set nthrds_ocn = 1
    set ntasks_cpl = 2; set nthrds_cpl = 1

if ($COMP_LND == xLnd) then
    set ntasks_lnd = 2; set nthrds_lnd = 1
endif

if ($COMP_ICE == xice) then
    set ntasks_ice = 4; set nthrds_ice = 1
endif

...
File “env.linux.newmach” (2 of 3)

```
# General machine specific environment variables - edit before the initial build

setenv EXEROOT /ptmp/$LOGNAME/$CASE
setenv RUNROOT $EXEROOT
setenv MAKE_J 1

# Environment variables for prestaging input data - edit anytime during run

setenv DIN_LOC_ROOT /fs/cgd/csm/inputdata
setenv DIN_LOC_ROOT_USER /fs/cgd/csm/inputdata_user
setenv DIN_LOC_MSROOT /CCSM/inputdata
setenv DIN_REM_MACH dataproc.ucar.edu
setenv DIN_REM_MSROOT /CCSM/inputdata
setenv DIN_REM_ROOT /fs/cgd/csm/inputdata
```

```
File "env.linux.newmachi" (3 of 3)

::

# Environment variables for short term output storage - edit anytime during run

setenv DOUT_S
setenv DOUT_S_ROOT

# Environment variables for longer term output storage - edit anytime during run

setenv DOUT_L_RCP
setenv DOUT_L_RCP_ROOT
setenv DOUT_L_MS
setenv DOUT_L_MSNAME
setenv DOUT_L_MSRoot
setenv DOUT_L_MSPWD
setenv DOUT_L_MSRPd
setenv DOUT_L_MSPRJ

::
newmach

#! /bin/csh -f

set mach = bangkok

source $CASEROOT/env_conf || exit -1
source $CASEROOT/env_run || exit -1
source $CASEROOT/env_mach.${mach} || exit -1

# Determine the nodes for batch queue
@ cpu_per_node = 2
@ tasks = 0
@ mpp = 0
foreach n (1 2 3 4 5)                   # loop over components
  @ tasks = $tasks + $NTASKS[$n]
  @ mpp = $mpp + $NTASKS[$n] * $NTHRDS[$n]
end

::

::
"batch.linux.newmach" (2 of 2)

```bash
# This is a CCSM batch job script for $mach

### BATCH INFO
#PBS -q medium
# Maximum number of processes (CHANGE THIS if needed)
#PBS -l nodes=$nodes:ppn=$cpu_per_node
# output file base name
#PBS -N testbatch.pbs
# Put standard error and standard out in same file
#PBS -j oe
# Export all Environment variables
#PBS -V
# End of options
```
# Create processor count input files

```bash

cd $EXEROOT/all
@ PROC = 0  # counts total number of tasks
rm -rf mpirun.pgfile1 # create new pgfile
rm -rf mpirun.pgfile  # create new pgfile
echo "0" >! mpirun.pgfile1;
foreach n (1 2 3 4 5)
    set comp = $COMPONENTS[$n]
    set model = $MODELS[$n]
    set nthrd = $NTHRDS[$n]
    set ntask = $NTASKS[$n]
    @ M = 0
    while ( $M < $ntask )
        if (($n == 1) && ($M == 0)) then
            echo "skipping first model"
        else
            echo "1 $EXEROOT/all/$comp" >! mpirun.pgfile1;
        end
    end
```
File “run.linux.newmach” (2 of 4)

```bash
::

# ---------------------------------------------------------------------------
# Run the model
# ---------------------------------------------------------------------------

cd $EXEROOT/all
paste ${PBS_NODEFILE} mpirun.pgfile1 > mpirun.pgfile
echo "date" -- CSM_EXECUTION BEGINS HERE
mpirun -p4pg mpirun.pgfile ./${COMPONENTS[1]}
wait
echo "date" -- CSM_EXECUTION HAS FINISHED

# ---------------------------------------------------------------------------
# Save model output stdout and stderr
# ---------------------------------------------------------------------------

cd $EXEROOT/cpl
set CplLogFile = `ls -1t cpl.log* | head -1`
::
```
File "run.linux.newmach" (3 of 4)

:::

# Perform short term archiving of output
# -----------------------------------------

if ($DOUT_S == 'TRUE') then
    echo "Archiving ccs3 output to $DOUT_S_ROOT"
    echo "In $CASEROOT directory using the short term archiving script ccs3_s_archive.csh"
    cd $CASEROOT; $UTILROOT/Tools/ccsm_s_archive.csh
endif

# Submit longer term archiver if appropriate
# -----------------------------------------

if ($DOUT_L_MS == 'TRUE' && $DOUT_S == 'TRUE') then
    echo "Long term archiving ccs3 output using the script $CASE.$MACH.l_archive"
    gsub $CASE.$MACH.l_archive
endif

:::
```
# Resubmit another run script
set echo
cd $CASEROOT
source env_run
if ($RESUBMIT > 0) then
    echo RESUBMIT is $RESUBMIT
    echo RESUBMIT = $RESUBMIT - 1
    echo RESUBMIT is $RESUBMIT
    sed '1/^ *setenv *CONTINUE_RUN .*/s//setenv CONTINUE_RUN TRUE/' \
        env_run > env_run.tmp; mv env_run.tmp env_run
    sed "s/^ *setenv "$RESUBMIT".*/setenv $RESUBMIT $RESUBMIT/;" \
        env_run > env_run.tmp; mv env_run.tmp env_run
    qsub $CASE.$MACH.run
endif
```
The Macros file

- Located in $CCSMROOT/models/bld
- Where you place the build specific modifications (do not change the Makefiles)
- Examples
  - Macros.Linux
  - Macros.AIX
ifeq \$(\text{MACH}), \text{jazz}
    \text{INCLDIR} := -I. \ -I/soft/apps/packages/netcdf-3.5.0/include \ -I/usr/include \ -I\$(\text{INCRoot}) \ -I/soft/apps/packages/mpich-gm-1.2.5.9-pre6-gm-1.6.3-pgi-4.0/includ
    \text{SLIBS} := -L/soft/apps/packages/netcdf-3.5.0/lib \ -lncdf \ -llapack \ -lblas
else
    \text{INCLDIR} := -I. \ -I$(\text{INC_NETCDF}) \ -I$(\text{INCRoot}) \ -I$(\text{INC_MPL})
    \text{SLIBS} := -L$(\text{LIB_NETCDF}) \ -lncdf \ -llapack \ -lblas
endif

::
::
File “Macros.Linux” (2 of 2)

```plaintext
.
.
ULIBS := -L$(LIBROOT) -lesmf -lmct -lmpeu -lmph
CPP := NONE
CPPFLAGS := -DLINUX -DPGF90 -DNO_SHR_VMATH
CPPDEFS := -DLINUX
CC := mpicc
CFLAGS := -c
ifeq ($(CC),pgcc)
    CFLAGS += -fast
else
    CFLAGS += -DUSE_GCC
endif
FIXEDFLAGS :=
FREEFLAGS := -Mfree
FC := mpif90
FFLAGS := -c -r8 -i4 -Kieee -Mrecursive -Mdalign -Mextend
.
.
```
Performance Tuning

- Default run configuration
  - Basic “will run” configuration

- You might want to change to
  - Optimize machine (cpu) efficiency
  - Optimize run speed
  - Reflect machine limits
  - Reflect usage options or new science
How Does It All Fit Together?

- The new machine files are used
  - When you execute "create_newcase"
    - Creates new case directory with configure, env_run, env_mach.newmach, env_conf
  - "configure" command creates
    - <casename>.newmach.build
    - <casename>.newmach.run
Gotcha’s

- Your MPI must support multiple binaries
  - Each component is a different binary
- Your search path needs to be able to find the mpi and compiler files
- A new processor and/or compiler may not generate correct results
- I/O can impact performance
Finalizing The Process

- Get the file additions and changes back to NCAR
  - If the machine is uniquely different and interesting we may be able to get it into a later release

- Provide performance numbers
  - We may be able to serve various performance numbers to the community
Plans For Future

- Simplification/Modularization for reuse
- Range of configuration selections
  - Optimal processor utilization
  - Optimal run speed
- How does one know what is optimal?
  - Somewhat complicated
  - Beginnings of scripts to help
    - ./scripts/ccsm_utils/Tools/timing/getTiming.csh
  - Web based information
Questions

- For more information see Section 6.10 of the User’s Guide

- ???