Cloud Locking in the Community Atmosphere Model, Version 6 (CAM6)

User Guide

*July 2020*

***Contributors***

Eleanor Middlemas (University of Colorado)

Jerry Olson (NCAR)

Jim Benedict\* (University of Miami and NCAR visiting scientist)

Brian Medeiros (NCAR)

*\* Affiliation as of Fall 2020: Los Alamos National Lab*

Table of Contents

Page

Preface 3

Step

0. Basic CESM user knowledge and specific CESM versions required 7

1. Prepare workspace with a copy of CESM, source code modifications, and cloud locking scripts v

2. Run a simulation that generates cloud fields that can be used for subsequent “cloud-locked” experiments vi

3. Prep local model sandbox so that cloud locking scripts are executed during cloud-locked simulation vi

4. Prestage cloud parameter files for cloud locking using post-run scripts vi

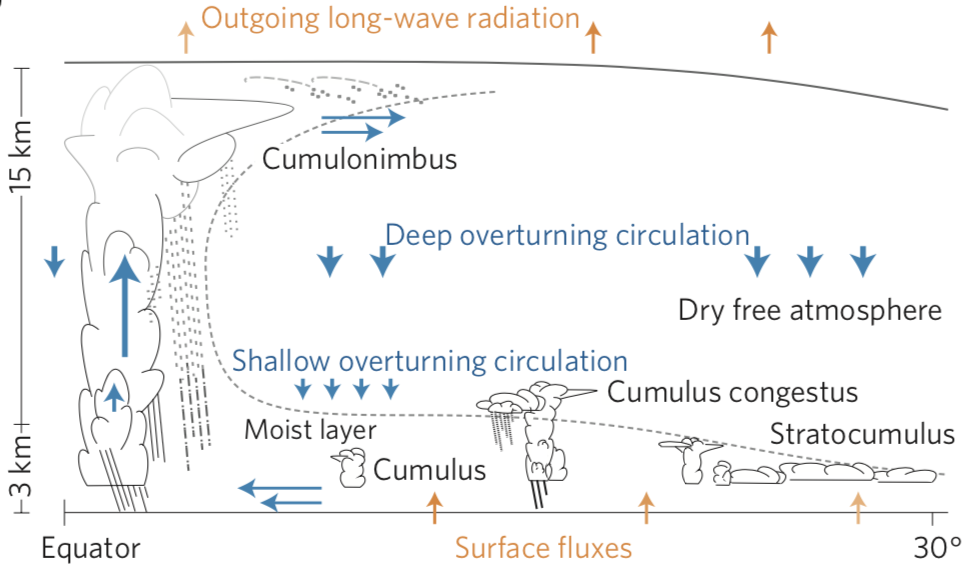
5. Run a cloud-locked simulation vi

6. Check to see if cloud locking worked vi

Appendix: Disk Space and Timing vi

Preface

**What is the purpose of cloud locking?**



*Image taken from Bony et al. (2015), Nature Geoscience.*

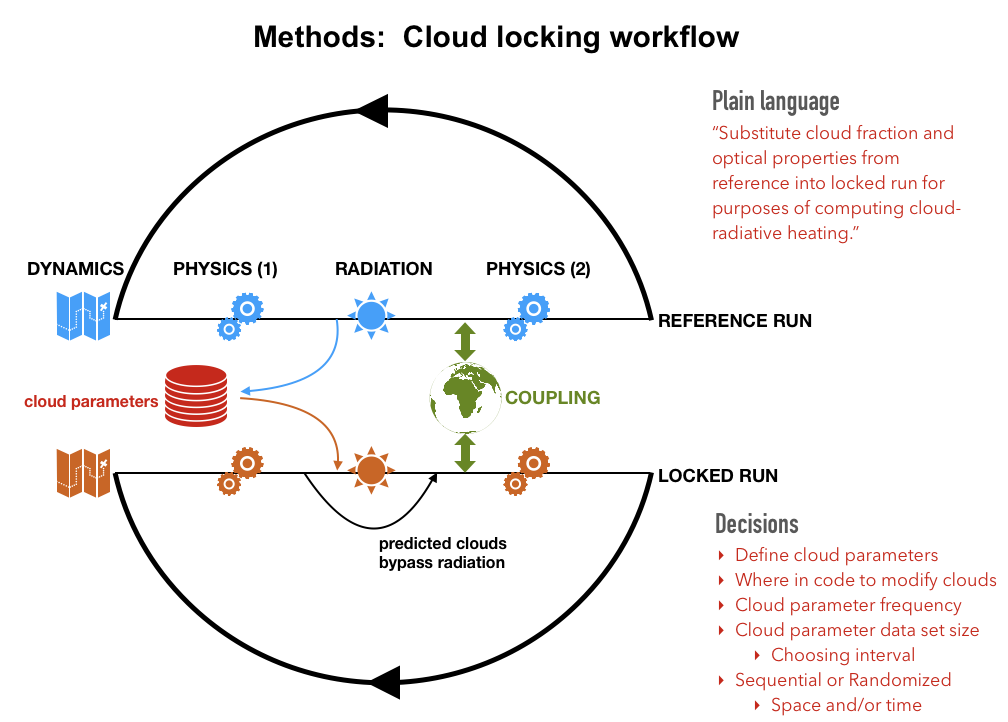
Clouds are intimately intertwined with Earth’s climate through atmospheric circulation. Clouds modulate the distribution of heat, momentum, and moisture, and play a critical role for Earth’s radiation balance. Clouds reflect incoming solar radiation, reducing the amount of energy reaching the surface (a cooling influence). They also absorb longwave radiation emitted from the surface, and radiate at a cooler temperature, effectively “trapping” energy in the system (a warming influence). Both cloud radiative effects impact the heating rates of a given atmospheric column, which couple to the atmospheric circulation, and thus creating a cloud-radiative feedback.

Given their strong radiative effects, clouds are key players in the climate response to external forcing. The Earth’s radiative balance is extremely sensitive to changes in clouds. The complexity of clouds, e.g., the range of scales from aerosol to cloud systems and their multi-phase composition, poses a challenge for representing clouds and their radiative properties in climate models.

Despite the difficulties in capturing clouds realistically, climate models provide a means of hypothesis testing and exploration in climate science. To advance understanding of cloud effects on climate and to help evaluate climate model fidelity, many methods have been employed to isolate and emphasize the roles that clouds play in the system. As noted, one way that clouds impact the system is through the direct coupling between cloud radiative effects and the circulation. The cloud locking method provides a way to separate cloud-radiative feedbacks from circulation by breaking that feedback.

Cloud locking prescribes cloud properties used as inputs to the radiation module of a given atmospheric model, essentially prescribing cloud radiative heating rates. As a result, the climate can evolve and produce some cloud changes, but independent cloud properties are used to determine the radiative effect on atmospheric circulation rather than the predicted cloud fields.

For example, in a cloud locking experiment, wind may carry some cloud over to a new gridbox. That cloud may precipitate. But, when it’s time for the model to calculate that cloud’s radiative properties, the model grabs the cloud for that gridbox from a file, rather than using the predicted cloud. That file’s cloud may actually be clear-sky. That particular gridbox cloud could contain less liquid and more ice or have less cloud fraction than the predicted cloud. In any case, the resulting cloud radiative forcing and associated heating rates would be different from the predicted clouds, ultimately radiatively decoupling clouds from the evolving climate system.



**Different methods of cloud locking: repeating years versus shuffling**

|  |  |  |
| --- | --- | --- |
|  | **Repeating years in cloud-locked simulation** | **Shuffling years in cloud-locked simulation** |
| Pros | Less computationally expensive | Removes autocorrelation of cloud-radiation interactions from timestep to timestep   * Ensures the effects of cloud feedback are removed on all timescales |
| Cons | Prescribing 1 year repeatedly may be prone to errors. For example, prescribe a particular stormy year, or an El Niño year, or a strong AMO/PDO year, or some other regional anomaly, etc  Autocorrelation on small timescales has been argued to impact longer timescales | Significantly more computationally expensive (both CPU hours and disk space)  More complicated methodologically, and possibly less reproducible as a result |
| Timestep in model simulation | Timestep from clouds that are repeated sequentially  (ex: 1 year is repeated) | Timestep from clouds that are randomly shuffled  (ex: sourced from a 3-yr data pool) |
| **Yr 0001** – Jan 01 – 00Z  **Yr 0001** – Jan 01 – 02Z  **Yr 0001** – Jan 01 – 04Z  **Yr 0001** – Jan 01 – 06Z  **Yr 0001** – Jan 01 – 08Z  .  .  .  **Yr 0001** – Dec 31 – 22Z  **Yr 0002** – Jan 01 – 00Z  **Yr 0002** – Jan 01 – 02Z  **Yr 0002** – Jan 01 – 04Z | **Yr 0001** – Jan 01 – 00Z  **Yr 0001** – Jan 01 – 02Z  **Yr 0001** – Jan 01 – 04Z  **Yr 0001** – Jan 01 – 06Z  **Yr 0001** – Jan 01 – 08Z  .  .  .  **Yr 0001** – Dec 31 – 22Z  **Yr 0001** – Jan 01 – 00Z  **Yr 0001** – Jan 01 – 02Z  **Yr 0001** – Jan 01 – 04Z | **Yr 0002** – Jan 01 – 00Z  **Yr 0003** – Jan 01 – 02Z  **Yr 0001** – Jan 01 – 04Z  **Yr 0001** – Jan 01 – 06Z  **Yr 0003** – Jan 01 – 08Z  .  .  .  **Yr 0003** – Dec 31 – 22Z  **Yr 0001** – Jan 01 – 00Z  **Yr 0002** – Jan 01 – 02Z  **Yr 0001** – Jan 01 – 04Z |

Debate exists about which cloud locking method is more appropriate. Some argue that prescribing one year repeatedly “locks” the clouds to a potentially anomalous annual cycle, and that there is a “basic” effect of clouds on smaller timescales that spread to longer timescales. Others argue that breaking the feedback between cloud radiative heating and climate is by far the biggest hammer (i.e., any type of cloud locking), no matter how one performs it.

The cloud-locking method precisely disables cloud-radiation interactions without directly altering the atmospheric state, and ensures similar climatologies between control and locked simulations.  However, multi-decadal cloud-locked simulations with some models exhibit a weak warming or cooling drift relative to the control run (e.g., Mauritsen et al. 2013; Rädel et al. 2016; Middlemas et al. 2019; Olonscheck et al. 2019, Li et al. 2020) that may be driven by a modest artificial radiative forcing that arises from the loss of spatio-temporal structure in clouds (Schneider et al. 1999; Langen et al. 2012).  Cloud-radiative heating in cloud-locked experiments using randomized data pools has no temporal autocorrelation (no "memory"); whether randomized or not, cloud-radiative processes are fully decoupled from the predicted dynamical state.

Step 0. Prerequisites and Preliminaries

1. Background: CESM is a climate model written in Fortran and run through a Unix-like command line interface, usually on a high-performance compute (HPC) platform. Basic knowledge of unix, including editing files, is needed.
2. Obtain CESM, which is publicly available via GitHub. Instructions and additional information are available on the CESM website: <https://www.cesm.ucar.edu>. In the examples, we will use CESM on NCAR’s Cheyenne system using the glade filesystem. All steps for cloud-locking should be independent of the system. See the CIME user guide for help on porting to new systems (<http://esmci.github.io/cime/versions/master/html/users_guide/index.html#case-control-system-part-2-configuration-porting-testing-and-use-cases>).
3. Other notes:
   1. This guide provides instructions for implementing cloud locking in CESM2 using CAM6 with RRTMG as the radiation parameterization. The implementation in CESM1 with CAM5 is very similar. A different set of code modifications is necessary for CAM4 using CAM-RT for radiation.
   2. Filenames, directories and their paths, and executable code in the command line will be in typewriter font. A dollar sign ($) prefix indicates a shell-like variable evaluation. For example, $WORD is replaced with the value of the WORD variable, $USERNAME indicates the current user’s username, and the same goes for filenames ($FILENAME) and paths ($PATH/TO/DIRECTORY).
   3. Example cases, scripts, and source modifications for CAM6 cloud locking are archived in /glade/work/eleanorm/cases/cesm2\_cldlck. Users may additionally find cloud locking-relevant edits to the cesm2.1.3 codebase (outlined in step 3) in /glade/work/eleanorm/models/cesm2\_1\_3
   4. Maybe a note about our github repo here…?
4. Some resources
5. Getting started on NCAR HPC: <https://www2.cisl.ucar.edu/user-support/new-user-orientation>
6. CESM tutorial: <http://www.cesm.ucar.edu/events/tutorials/>
7. CESM quick start guide: <https://escomp.github.io/CESM/release-cesm2/>
8. User manual: <https://github.com/ESMCI/cime/wiki/CIME-Users-Guide>
9. An understanding of run types, i.e., hybrid, branch, and startup, is recommended (<https://github.com/ESMCI/cime/wiki/Customizing-runtime-settings>)

Step 1. Prepare workspace with a copy of CESM, source code modifications, and cloud locking scripts

1. Checkout a copy of CESM version 2.1 or later that runs CAM6. Instructions on how to do so are here: <https://escomp.github.io/CESM/release-cesm2/downloading_cesm.html>

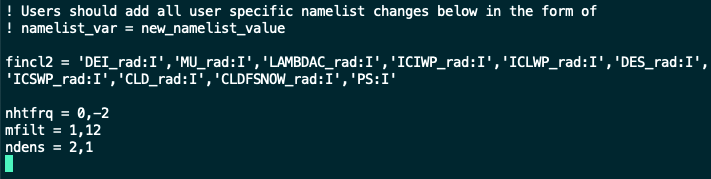
*\* Test running one or more scientifically validated compset(s) with the newly aquired version of CESM following instructions for creating a new case in the quick start guide (link referenced in part 0, #3).*

1. Copy cloud locking source code modifications to a local directory accessible to the user. https://github.com/NCAR/cesm\_cloud\_locking
   1. These source code modifications tell the model to either output radiative cloud parameters or read in those cloud parameters from a file.
   2. The source modifications include:
      1. ***cloud\_rad\_props.F90***: additional lines that read in a logical type variable prescribed\_cloud. If true, retrieves cloud parameters with suffix “\*\_in” from the physics buffer (“pbuf”) instead of the predicted cloud parameters. The pbuf can be thought of as a bag full of variables that the model uses during its computation. When a variable is needed for computation, the variable must be grabbed from the bag (read from the pbuf).
      2. ***namelist\_definition.xml***: adds the definition of namelist variables that can be referenced in the user\_nl\_cam file so the user may utilize cloud locking. These namelist variables include a path to a directory that contains symbolic links pointing to cloud parameter files (prescribed\_cloud\_datapath), as well as the name of the first symbolic link file pointing to the first cloud parameter file to be used for locking (prescribed\_cloud\_file).
      3. ***physpkg.F90***: this module drives the physics of CAM. The primary function of the modifications to physpkg.F90 involve registering, initializing, and advancing prescribed\_cloud.F90. By adding flags for prescribed\_cloud, this enables CAM to treat the prescribed cloud parameters like any other tracer variable.
      4. ***prescribed\_cloud.F90***: module (by Jerry Olson), which reads cloud-related fields and puts them into the physics buffer for use by the radiation scheme.
      5. ***radiation.F90***: CAM interface to RRTMG. Cloud parameters intended for output/input prior to the radiation tendency calculations are defined with the suffix “\*\_rad” and initialized. These are fields that are output straight from the radiation module and are used as the prescribed fields in a subsequent cloud locking experiment. Additionally, if cloud parameters are prescribed as in a cloud-locked simulation, the prescribed fields sourced from an external file are read in (with the suffix “\*\_rad”) from the pbuf.
      6. ***restart\_physics.F90***: uses prescribed\_cloud module to read and write restart files including the prescribed cloud fields.
      7. ***runtime\_opts.F90***: reads user-defined prescribed-cloud or cloud-locked namelist parameters
      8. ***tracer\_data.F90***: turns off interpolation so that when cloud parameters are read in, they are used in computations exactly as is from the file. This modification is essential for microphysics to be locked, because otherwise, the model will try to interpolate input files, even if input files are on the correct grid, which will cause some zero values to become non-zero, ultimately causing the model to crash.
   3. In any of these files, searching for prescribed\_cloud or cloud\_rad might help locate the cloud locking-related modifications.
2. Copy scripts that organize and stage the prescribed cloud property files “offline”, using either a random shuffling approach (i.e., to remove temporal autocorrelation) or a temporally sequential approach (read the cloud parameters in sequence and repeat selected year(s)).
   1. cp /glade/work/eleanorm/cases/cesm2\_cldlck/scripts/\* $USERCLDLCKSCRIPTSDIR
   2. The scripts to copy over include (i & iii are most important):
      1. adjustCloudData
      2. cloudLockResetDate.ncl
      3. sequenceCloudData.ncl
   3. The usage of these scripts will be outlined in more detail in steps 3 & 4.

Step 2. Run a simulation that generates cloud fields that can be used for subsequent “cloud-locked” experiments

1. Some things to note when setting up the simulation:
   1. To isolate the role of disabling cloud radiative feedbacks through cloud locking, users should be careful to use identical initial conditions and forcing between control and cloud locking simulations. Thus, it’s important that users save the restart files used for the control simulation as well as use the same CESM compset between the two simulations. Note: previous CAM users have had issues branching cloud-locked experiments. Instead, hybrid runtype is typically used.
   2. Typically, this simulation would be branched from whatever control simulation is used in the study.
   3. As noted in the appendix, a year of 3D 2-hourly CAM6 (0.9°x1.25° finite volume dynamical core) cloud fields requires **approximately** **267 GB** of disk space. Users should ensure they have the appropriate disk space.
2. Create a new case for outputting cloud parameters. Here, we shall call it $CLDOUTPUTCASEDIR.
   1. If changing the efficiency and speed of the simulation is desired, one may alter the PE layout (via env\_mach\_pes.xml). This is the time when one should do so. More information on changing the PE layout can be found here: <https://github.com/ESMCI/cime/wiki/How-to-set-up-a-case-and-customize-the-PE-layout>
   2. Then, run ./case.setup
3. Copy source code modifications into the user’s local case directory’s source modifications folder
   1. cp $USERSRCMODDIR/\* $CLDOUTPUTCASEDIR/SourceMods/src.cam/
4. Specify CAM namelists such that CAM will output instantaneous (“:I”) 2-hourly cloud radiative fields
   1. Add the following fields to $CLDOUTPUTCASEDIR/user\_nl\_cam  
      fincl2 = 'DEI\_rad:I','MU\_rad:I','LAMBDAC\_rad:I','ICIWP\_rad:I','ICLWP\_rad:I','DES\_rad:I','ICSWP\_rad:I','CLD\_rad:I','CLDFSNOW\_rad:I','PS:I'  
      nhtfrq = 0,-2  
      mfilt = 1,12

ndens = 2,1

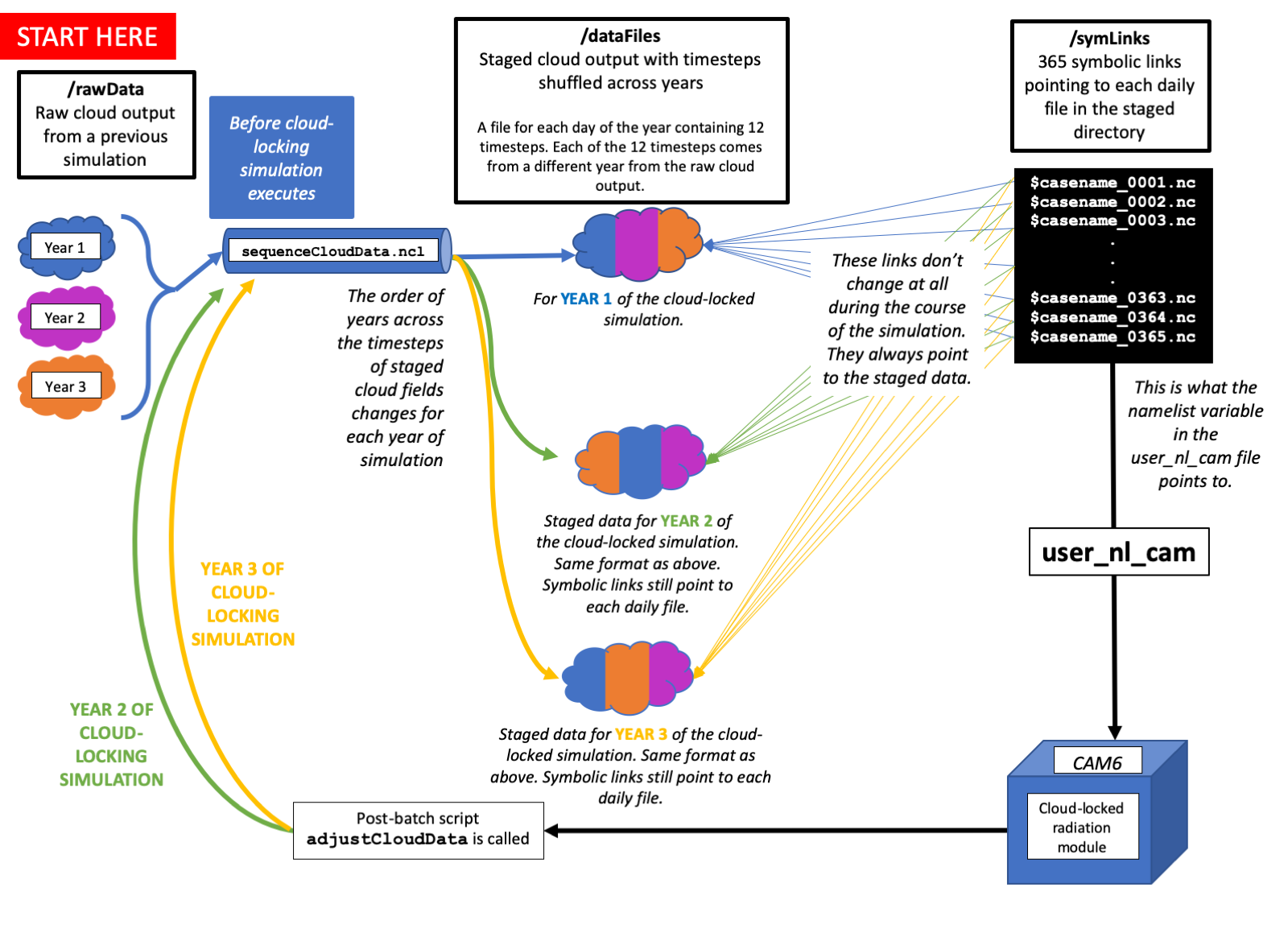


*An example of a user\_nl\_cam file for the experiment that outputs radiative cloud parameters.*

* 1. Explanations of those changes:
     1. ***fincl2*** specifies that, in addition to the default output or “history” file containing default CAM output, the listed variables should appear in an additional auxiliary history file. These are the cloud parameters that will be used for cloud locking in a subsequent experiment.
        1. The output will appear in the user’s run archive under the CAM folder. The default output will appear in files containing h0 and the cloud locking parameters will appear in files containing h1.
        2. The :I following each cloud parameter name tells the model to output the cloud fields instantaneously, rather than as an average over the period defined by nhtfrq below.
        3. The user may also specify additional output variables at the default monthly output frequency by defining fincl1 with variables not found in the default history files. Furthermore, users may output at another frequency by defining a variable fincl3 in a similar manner. These output files will have names containing h2.
     2. ***nhtfrq*** defines the frequency at which the listed variables will be output in number of hours x -1. The first number in the list is for the default history file (though it is not recommended to change this frequency to anything except for monthly). The second number in the list corresponds to the variables listed in fincl2. Here, I have specified that cloud parameters should be written out every 2 hours.
     3. ***mfilt*** defines the number of timesteps written per output file. I have written mine in such a way that every file has a 1 model day of output, or 12 2-hourly timesteps.
     4. Finally, ***ndens*** specifies the precision at which the output should be written. The default is that output is written in single precision (indicated by a “2”), but for cloud locking experiments, the cloud parameters should be output in double precision (indicated by a “1”).
     5. For further explanation of these variables, please see: <http://www.cesm.ucar.edu/models/cesm2/settings/current/cam_nml.html>

1. Make xml changes: edit $OUTPUTCASEDIR/env\_run.xml and $OUTPUTCASEDIR/env\_batch.xml files appropriately
   1. **Critical:** Currently, the cloud locking setup is designed such that a single year of prescribed cloud data is staged. Use of a single year of staged cloud data minimizes disk space requirements. If randomly shuffled prescribed cloud data are being staged, the user **must** submit one-year batch job requests to the NCAR supercomputer in order for the setup to work properly. In the case of random shuffling, the time window of the staged cloud data must equal or exceed the time length of the batch job. The staged cloud data window could be expanded to multiple years (thus allowing multi-year batch jobs), but this would require significant modifications to the scripts provided and has not yet been attempted or tested by the developers.
   2. Choose an appropriate wall clock time
   3. Choose the appropriate run type & change associated reference case files
   4. Specify run length
   5. Specify the frequency of restart files
2. Build and run the model
   1. Build: qcmd -- ./case.build
      1. Note: To run the above command, users will need to (a) have a CISL-provided project code (the same project number being charged to make the simulation is fine) -and- (b) ensure that environment variable PBS\_ACCOUNT is set to that project number.
   2. ./case.submit

*Schematic for cloud-shuffling*



Step 3. Prep the local model sandbox so that cloud locking scripts are executed during the cloud-locked simulation

***\* This step only has to be performed once per sandboxed CESM version. Users may skip this step when conducting future cloud locking experiments with the same CESM sandbox version. If a user’s sandbox version is deleted, this step must be repeated.***

***What is the function of these scripts?***

Upon completion of a given year during the cloud-locked experiment, a python batch script (“adjustCloudRun”) is called that prepares the prescribed cloud data for the next year of simulation and, if desired, shuffles the cloud data into the “staging area”. It does so by calling an NCL script called sequenceCloudData.ncl.

If shuffling, sequenceCloudData.ncl randomly sequences prescribed cloud fields immediately after the completion of the CESM batch job, i.e., 1 year of cloud locking simulation.  This "post-run" python script is not parallelized and requires over 1 hour of run time on a single Cheyenne node.

If the user decides to prescribe clouds in sequence, this script will simply redefine the dates within the prescribed-cloud files in the staging area such that they contain the date corresponding to the upcoming year in the simulation.

Because the post-run script handles the resubmission of subsequent CESM batch jobs, the next CESM batch job will not be submitted until the post-run script successfully completes.  This post-run script setup is modeled after the short-term archive script used to organize CESM output. Special thanks to Jim Edwards of NCAR's software engineering group!

The required steps to set up the post-run script in "batch" mode are:

1. **Define a FINAL version of the post-run python script, adjustCloudData.** This script directs the model to run the sequenceCloudData.ncl script, which prepares the staged cloud files for the next simulation year. It is recommended that the user retain a copy of this script in a local directory. The final version should contain the user-defined path to the sequenceCloudData.ncl script. The suffix “.py” of adjustCloudData should not be included, per instructions from CESM software engineers. This script already includes some lines required for a batch job submission to Cheyenne. The main tasks of the script should reside in a python function, not the main program.
2. **Copy FINAL version of post-run python script to "template" file.**  Once #1 above is complete, the adjustCloudData script needs to be copied to a specific “template” file (below) accessible by the model so that the script will be executed upon model completion. The user must create a "template" version of the post-run script and save it in a new file $CESMROOT/cime/config/cesm/machines/template.adjustCloudData. This template file must include some "batch system directives" (again, for proper sharing of information between CESM and the external post-run script). Specifically, the following lines should be included beginning on the second line of the post-run script:

# Batch system directives  
{{ batchdirectives }}  
import sys, os, time  
os.chdir( '{{ caseroot }}')  
  
\_LIBDIR = os.path.join("{{ cimeroot }}", "scripts", "Tools")  
sys.path.append(\_LIBDIR)

1. **Create POSTRUN\_SCRIPT\_BATCH logical variable to be added to env\_run.xml.**  In $CESMROOT/cime/src/drivers/mct/cime\_config/config\_component\_cesm.xml, add the following lines:

<entry id="POSTRUN\_SCRIPT\_BATCH">  
    <type>logical</type>  
    <valid\_values>TRUE,FALSE</valid\_values>  
    <default\_value>FALSE</default\_value>  
    <group>external\_tools</group>  
    <file>env\_run.xml</file>  
    <desc>Logical to invoke external script to be run as batch job after model completion</desc>  
</entry>

These lines must be added within the main <entry\_id version="3.0"> block.

These lines create a logical that can be set to run a post-run batch script or not, following successful completion of a CESM job submission.  It will also be used to determine whether CESM or the external post-run script performs the resubmission (if RESUBMIT > 0).

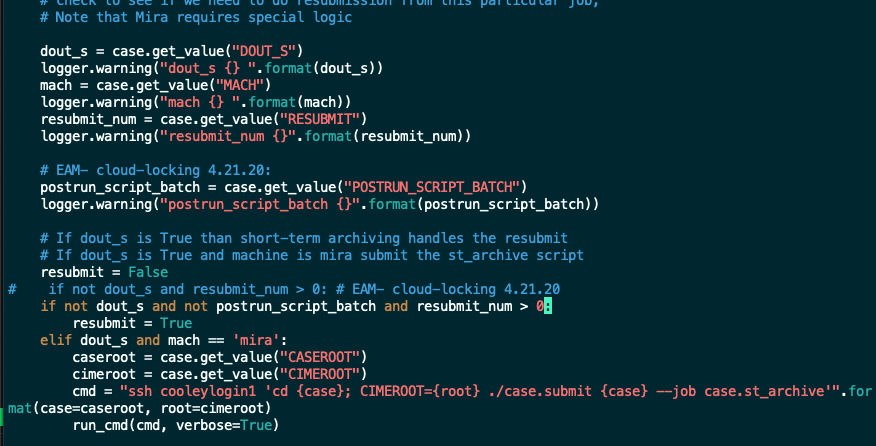
This newly defined logical POSTRUN\_SCRIPT\_BATCH will live in env\_run.xml. To change POSTRUN\_SCRIPT\_BATCH, simply use xmlchange in the usual way:  
./xmlchange POSTRUN\_SCRIPT\_BATCH="TRUE"

Default setting of POSTRUN\_SCRIPT\_BATCH is FALSE.

1. **Slightly modify the way that CESM job resubmissions are handled.**  If POSTRUN\_SCRIPT\_BATCH is TRUE, we must prevent case\_run.py from resubmitting the next CESM batch job and instead have the post-run script do the resubmit.  This is akin to disabling resubmit in case\_run.py when using short-term archiving. Make the following changes to $CESMROOT/cime/scripts/lib/CIME/case/case\_run.py:
   * Near the top of the \_resubmit\_check function, just below the retrieval of dout\_s and resubmit\_num, add the following to retrieve POSTRUN\_SCRIPT\_BATCH logical:

postrun\_script\_batch = case.get\_value("POSTRUN\_SCRIPT\_BATCH")  
logger.warning("postrun\_script\_batch {}".format(postrun\_script\_batch))

* + A few lines below this, revise the if-statement as follows:  
    OLD:  if not dout\_s and resubmit\_num > 0:  
    NEW:  if not dout\_s and not postrun\_script\_batch and resubmit\_num > 0:



*An example of a user’s updated case\_run.py file such that the model utilizes the user defined logical POSTRUN\_SCRIPT\_BATCH flag.*

1. **Inform CESM of the default post-run batch job characteristics.**

*This change is different between CESM2.0 and CESM2.1. The following change is written for CESM2.1.*

In $CESMROOT/cime/config/cesm/machines/config\_workflow.xml, in the <workflow\_jobs id="default"> section at bottom add the following (analog of the short-term archiver):

<job name="case.adjustCloudData">

<template>template.adjustCloudData</template>

<!-- If case.run exits successfully then run adjustcloudData-->

<dependency>case.run</dependency>

<prereq>$POSTRUN\_SCRIPT\_BATCH</prereq>

<runtime\_parameters>

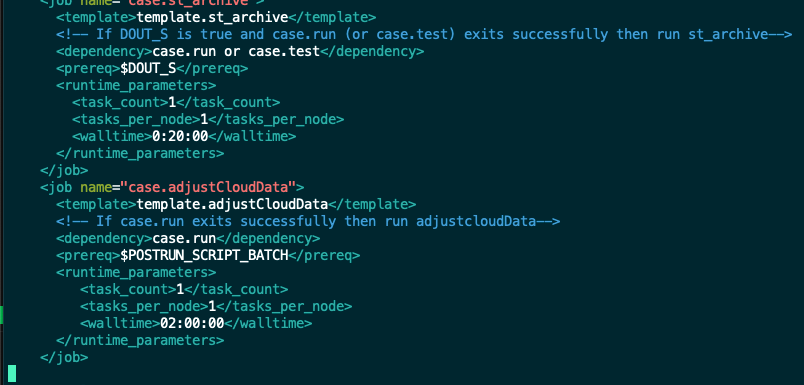
<task\_count>1</task\_count>

<tasks\_per\_node>1</tasks\_per\_node>

<walltime>02:00:00</walltime>

</runtime\_parameters>

</job>

****

*An example of a user’s updated config\_workflow.xml file such that the adjustCloudData python script is executed at the completion of every CESM batch job.*

1. **Load modules that are used by the post-run script into the CESM environment.** Because the sequenceCloudData script is written in NCL, the NCL module must be properly loaded within the CESM "environment" (it is not by default).  In $CASEROOT/cime/config/cesm/machines/config\_machines.xml, add the following line, in bold & italics, to this section:

<machine MACH="cheyenne">  
  <module\_system type="module">  
    <modules compiler="intel">  
      ***<command name="load">ncl/6.6.2</command>***  
    </modules>  
  </module\_system>  
</machine>

The user may see other load commands in the same section.

If another model compiler is used (for example, if the user has ported CESM onto a different system), the user would need to add the "load" line to other compiles (pgi, gnu) as needed.

**Notes:**

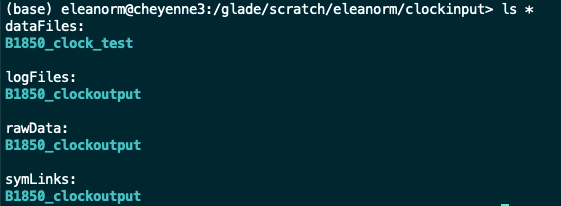
* For #1 & #2, if any changes are made to the version of the post-run script NOT located in $CESMROOT/cime/config/cesm/machines, those changes MUST be propagated into the "template" file. So long as the sequenceCloudData.ncl script does not move locations, the user should not have to update adjustCloudData in his/her local scripts directory nor in the cesm sandbox folder. Note that sequenceCloudData.ncl can be modified by the user without the need to modify adjustCloudData.
* To test if there are typos, one may try to create and run a new case for a very short (<5 days) period

Step 4. Prestage cloud parameter files for cloud locking using post-run scripts

***For non-shuffled simulations***: The CESM cloud locking simulation executes for 1 year. If the 1-yr run finishes without errors, adjustCloudData.py is called, adjustCloudData.py calls cloudLockChangeDate.ncl, and cloudLockChangeDate.ncl increments the 'date' year stamp by 1 in each cloud data file. No changes to the symbolic links in the symLinks directory are necessary. Once adjustCloudData.py finishes, the next CESM 1-year job is submitted and the process repeats.

***For shuffled simulations:*** The CESM cloud locking simulation executes for 1 year. If the 1-yr run finishes without errors, adjustCloudData.py is called, and adjustCloudData.py calls sequenceCloudData.ncl. This script re-sequences the staged (prescribed) cloud data in dataFiles/ by retrieving, for each sub-daily time step and each calendar day (to preserve the diurnal and seasonal cycles), cloud properties from a randomly selected year within the data pool (rawData/ directory). Thus, a **single year (365 files)** of staged cloud data files exist in this directory. Names of these files are not important; what is important is that the year stamps of the date variable have been rolled back to "1" (if the cloud locking run is started from Year 1).  No changes to the symbolic links in the symLinks directory are necessary. Once adjustCloudData.py finishes, the next CESM 1-year job is submitted and the process repeats. This reshuffling takes around an hour to complete on Cheyenne.

1. Set up file directories for cloud locking:
   1. Staging directory (dataFiles/)
   2. Symbolic link directory (symLinks/). Static symlinks are contained in a separate directory and are named sequentially (\*\_00001, \*\_00002, \*\_00003, …, \*\_00367). Each link points to the corresponding daily file in the dataFiles/ directory.
   3. Directory for script log files (logFiles/)
   4. If desired, a directory for a copy of the raw cloud parameter output data (rawData/). If the user choses to have this directory, this is where 2-hourly data outputted from Step 2 should go. Otherwise, the user should simply take note of where his/her outputted 2-hourly cloud data lives and set dirRawDataPool in sequenceCloudData.ncl accordingly, Note that sequenceCloudData.ncl is currently set to use **all** available cloud output within dirRawDataPool for the data pool—if contains 3 yr of 2-hourly cloud output, the data pool will span 5 yr as well.
   5. Within each directory, it is recommended that there should be another directory for each corresponding cloud locking experiment. Doing so facilitates data organization and ensures that a “paper trail” of the prescribed-cloud sequencing order used in the cloud-locked runs is archived in case the simulation ever needs to be reproduced. For example, the names of these directories could be the same as the user’s cloud-locking experiment casename. Take note of these directories, as their paths will be used in sequenceCloudData.ncl. (settings dirProcData and procDataCaseName)



*Eleanor’s example file directory structure for cloud-locking*

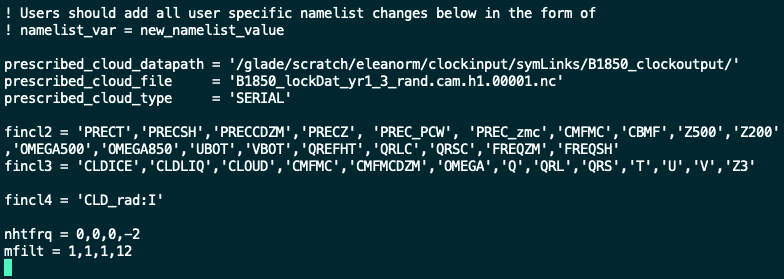
1. After updating the user-defined section with user-defined directory paths, submit sequenceCloudData.ncl to a node on Casper or Cheyenne to run once prior to starting the cloud locking simulation. This “stages” raw cloud data in the appropriate folder. On Casper, this took about 8-10 hours.
2. Check the date and datesec within the input files to make sure they’re monotonically increasing and matching those of the run defined by env\_run.xml. A good way to do this for the 01-Jan file is:  
     
   ncdump -v date,datesec /path/to/symLinks/\*00001.nc   
     
   The date and datesec combination in symlink file \*00001.nc should match the expected starting point of the cloud-locked run. For example, if the user is just beginning a cloud-locking simulation, date in /path/to/symLinks/\*00001.nc should be consistent with RUN\_STARTDATE in env\_run.xml and datesec =0 should be included in that file. For subsequent job submissions, date in /path/to/symLinks/\*00001.nc should match the expected year, month, and day of the model’s restart time. It is recommended that the user also check date and datesec in the file corresponding to the final day of prescribed data, /path/to/symLinks/\*00365.nc.

**Notes:**

* A helpful tool is script cloudLockResetDate.ncl, which gives the user the power to manually modify the year of the staged cloud data files. This may be useful if mistakes are made during the user’s initial manual execution of sequenceCloudData.ncl (note that each time sequenceCloudData.ncl is called, the year will be incremented forward by 1. This code may also be used in subsequent cloud locking simulations to set the cloud dates to an arbitrary year. The user will also need to change the paths listed in cloudLockResetDate.ncl to point to their data directories accordingly.
* The files names in dataFiles/CASE1/ are arbitrary.  The model directly reads the "files" in symLinks/CASE1/ which have integer file name stamps and simply point to the cloud data files under dataFiles/
* I found it helpful to keep the prescribed cloud data within dataFiles/, symLinks/, and logFiles/ separate for each individual locked run, hence the "CASE1", "CASE2", etc. subdirs.  This allows multiple locked runs to occur simultaneously, and permits extending runs. A disadvantage: it takes up more disk space.
* Note that there are 367 symbolic links files in symLinks/CASE1.  This is because if the model is initialized at 00Z 0001-01-01, it requires (but does not actually use?) prescribed cloud data from 0000-12-31 as well as 0002-01-01.

Step 5. Run a cloud-locked simulation

1. Create new case for cloud locking simulation. We shall call it $CLDLOCKCASEDIR.
   1. Again, if changing the efficiency and speed of the simulation is desired, one may alter the PE layout (via env\_mach\_pes.xml). This is the time when one should do so. More information on changing the PE layout can be found here: <https://github.com/ESMCI/cime/wiki/How-to-set-up-a-case-and-customize-the-PE-layout>
   2. Then, run ./case.setup
2. Copy source code modifications into the user’s local case directory’s source code modifications folder
   1. cp $USERSRCMODDIR/\* $CLDLOCKCASEDIR/SourceMods/src.cam/
   2. This is also where the user may edit the cloud locking source code modifications such that cloud locking occurs only in a specific region. To do so, the user would search for EAM - locking to find a line where a user has locked in the Arctic previously. Latitudes & longitudes are defined by clat and clon, respectively, and are expressed in degrees.
3. Edit $CLDLOCKCASEDIR/user\_nl\_cam file
   1. Add CAM namelist variables such that the model knows where to find symbolic links created in Step 4. These variables were defined in the source modifications file namelist\_definition.xml.
      1. prescribed\_cloud\_datapath = $PATH/TO/SYMLINKFOLDER
      2. prescribed\_cloud\_file = $NAMEOFFIRSTSYMLINKFILE
      3. prescribed\_cloud\_type = 'SERIAL'
   2. This is where the user would choose to run COSP and change output file frequency and content.



*Example user\_nl\_cam file for cloud locking simulation. Note the first three variables defined are essential for cloud locking. This example also includes the option to output CLD\_rad instantaneously every 2 hours as a diagnostic tool for checking whether cloud locking is working (Step 6).*

1. Change xml variables accordingly.
   1. ***Important: As it is set up in the provided scripts, cloud-locking jobs must be 1 year in length (e.g.,* STOP\_N=12 *for* STOP\_OPTION=”nmonths” *or* STOP\_N=1 *for* STOP\_OPTION=”nyears” *in env\_run.xml! More generally, job submissions for cloud locking simulations must be equal to or (in very rare circumstances) less than the time span of the staged cloud data files, otherwise the model will crash because it will look for a time stamp that does not exist. Again, the scripts provided only support staged cloud data files spanning 1 year and so the requested simulation job should also be 1 year in length.***
   2. Don’t forget to run ./xmlchange POSTRUN\_SCRIPT\_BATCH="TRUE" so that our cloud locking scripts will execute.
   3. Otherwise, change in a similar fashion to the cloud output run.
   4. Ensure that initialization is identical to cloud output simulation from Step 2.

**Notes:**

This is by far where the most debugging occurs, especially when shuffling cloud fields. The model is really picky about the dates & times of the cloud parameter files. Expect that one may have to repeat Step 4 a few times to redefine cloud file dates and timesteps.

Step 6. Check to see if cloud locking worked

* The most robust check: If instantaneous 2-hourly cloud radiative parameter files in the cloud-locked simulation are written to file (see user\_nl\_cam example in Step 4), one can directly compare this 2-hourly output to the corresponding 2-hourly prescribed-cloud-property inputs—they should be identical (taking care to match the dates and time correctly can be a little tedious).
* Compare some month of monthly-averaged cloud radiative forcing from cloud-output simulation to cloud-locked simulation… this will provide a qualitative check but will not fully confirm that cloud locking is working.
* Another qualitative check: If more than a year (for prescribing one year repeatedly) or if less than a year (for randomized shuffling), the user can compute the regression of cloud radiative forcing on any given variable as a metric of “cloud radiative feedback”. Note: this metric is subject to cloud masking, i.e., cloud forcing is not the best metric for cloud radiative feedback (though, the best metric is somewhat subjective…)

Appendix: Disk Space and Timing

Cloud locking input file organization (for shuffling three years of 2-hourly cloud parameters):

~1.2T of data

732 mb per day of cloud parameter files (i.e., 267180 mb or 267.18 gb per year)

**Cloud output run (no cloud locking. essentially a control simulation):**

**Timing**

With my env\_mach\_pes.xml layout (from Jim Benedict)

Overall Metrics:

Model Cost: 3693.43 pe-hrs/simulated\_year

Model Throughput: 16.38 simulated\_years/day

Init Time: 67.513 seconds

Run Time: 15366.404 seconds 14.456 seconds/day

**Cloud locking run:**

**Timing**

Overall Metrics:

Model Cost: 6396.55 pe-hrs/simulated\_year

Model Throughput: 9.46 simulated\_years/day

Init Time: 179.776 seconds

Run Time: 9137.924 seconds 25.035 seconds/day

Final Time: 0.084 seconds