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

CLM2.1 User’s Guide

Mariana Vertenstein, Keith Oleson, Sam Levis and Peter Thornton
January 2003
Contents

1 What is new with CLM2.1 2

2 Obtaining the Source Code and Datasets 2

3 Creating and Running the Executable 4
   3.1 offline mode: using jobscript.csh 5
      3.1.1 Specification of script environment variables 5
      3.1.2 Setting the Namelist 6
      3.1.3 Creation of header and directory search path files 6
      3.1.4 Building the model 8
      3.1.5 Running the executable 8
   3.2 cam mode 8
   3.3 cosm mode 9

4 Namelist Parameters 9
   4.1 Specification of run length, run type and initial run date 9
   4.2 Specification of model input datasets 11
   4.3 Specification of history and restart files 14
   4.4 Specification of input physics variables 18
   4.5 Specification of RTM River routing 18
   4.6 Specification of cam mode namelist 19

5 CLM2.1 Data Structures 19

6 History File Fields 20
   6.1 Model history fields 20
   6.2 Adding new history fields 23

7 Offline Mode Namelist Examples 27
   7.1 Example 1: Offline initial run, one day, global 27
   7.2 Example 2: Restart run 29
   7.3 Example 3: Branch run 29
   7.4 Example 4: Auxiliary history files 30
   7.5 Example 5: Generation of regional grid surface dataset 31
   7.6 Example 6: Generation of global gaussian surface dataset 32

8 Testing Model Changes 33
1 What is new with CLM2.1

CLM2.1 corresponds to the incorporation of totally new hierarchical subgrid data structures into the land model code and produces only roundoff level changes when compared to CLM2.0.

In what follows, we provide a brief summary of the new CLM2.1 data structures. The subgrid hierarchy in CLM2.1 is composed of gridcells, landunits, columns and plant functional types (pfts). Each gridcell can have a different number of landunits, each landunit can have a different number of columns and each column can have multiple pfts. This results in efficient memory allocation, and allows for the implementation of many different types of subgrid representation.

The first subgrid level, the landunit, is intended to capture the broadest spatial patterns of subgrid heterogeneity. These broad patterns include the physically distinct surface types that were treated as special cases in the previous versions of CLM2.0 (e.g. glaciers and lakes). In terms of CLM2.0 variables, the central distinguishing characteristic of the landunit subgrid level is that this is where physical soil properties are defined: texture, color, depth, pressure-volume relationships, and thermal conductivity. In CLM2.1, landunits are used to represent the special landcover types (e.g. glacier and lakes), with a single additional landunit for the gridcell vegetated area.

The second subgrid level, the column, is intended to capture potential variability in the soil and snow state variables within a single landunit. The central characteristic of the column subgrid level is that this is where the state variables for water and energy in the soil and snow are defined, as well as the fluxes of these components within the soil and snow. Regardless of the number and type of pfts occupying space on the column, the column physics operates with a single set of upper boundary fluxes, as well as a single set of transpiration fluxes from multiple soil levels. These boundary fluxes are weighted averages over all pfts.

The third and final subgrid level is referred to as the plant functional type (pft), but it also includes the treatment for bare ground. It is intended to capture the biophysical and biogeochemical differences between broad categories of plants, in terms of their functional characteristics. All fluxes to and from the surface are defined at the pft level, as are the vegetation state variables (e.g. vegetation temperature, canopy water storage, and carbon and nitrogen states for the leaf, stem, and roots).

In addition to state and flux variable data structures for conserved components at each subgrid level (energy water, carbon, nitrogen, etc.), each subgrid level also has a physical state data structure for handling quantities that are not involved in conservation checks (diagnostic variables). For example, soil texture is defined through physical state variables at the landunit level, the number of snow layers and the roughness lengths are defined as physical state variables at the column level, and the leaf area index and the fraction of canopy that is wet are defined as physical state variables at the pft level.

This subgrid hierarchy is implemented in CLM2.1 as a set of nested derived types. Extensive use is made of pointers, both for dynamic memory allocation and for simplification of the derived type referencing within subroutines. The use of pointers for dynamic memory allocation ensures that the number of subgrid elements at each level in the hierarchy is flexible and resolved at run time, thereby eliminating the need to statically declare arrays of fixed dimensions that might end up being sparsely populated. The use of pointers for referencing members of the derived data type within the subroutines provides a coherent treatment of the logical relationships between variables (e.g., the user cannot inadvertently change a pft-level variable within a subroutine that is supposed to operate on the column states and fluxes), and a more transparent representation of the core algorithms (it is easy to tell when the code is in a column or pft loop).

2 Obtaining the Source Code and Datasets

The source code and datasets required to run the Community Land Model version 2.1 (CLM2.1) in offline mode (uncoupled from other components of the Community Climate System Model version 2 (CCSM2.0.1)) can be obtained via the web from:

http://www.cgd.ucar.edu/tss/clm
The user should refer to the CAM2.0 User’s Guide or the CCSM2.0.1 General Documentation for instructions on obtaining code and datasets to run CLM2.1 coupled to other CCSM2.0.1 components. It is assumed that the user has access to the utilities tar, Free Software Foundation gunzip and gmake (GNU gmake).

The CLM2.1 distribution consists of two tar files:

CLM2.1_code.tar.gz

and

CLM2.1_inputdata.tar.gz.

The file CLM2.1_code.tar.gz contains code, documentation, and scripts. This file must first be uncompressed with the gunzip utility and then "untarred" as follows:

```
gunzip -c CLM2.1_code.tar.gz | tar xvf -
```

The above command both uncompresses and "untars" the code into a clm2/ subdirectory. The directory hierarchy for "clm2/" is as follows:

```
Table 1: Source Code Directory Structure

<table>
<thead>
<tr>
<th>Directory Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/</td>
<td>Directory of FORTRAN and &quot;C&quot; source code</td>
</tr>
<tr>
<td>src/biogeophys/</td>
<td>Biogeophysics routines (e.g., surface fluxes)</td>
</tr>
<tr>
<td>src/camclm_share/</td>
<td>Code shared between the CLM2 and CAM (e.g., calendar information)</td>
</tr>
<tr>
<td>src/csm_share/</td>
<td>Code shared by all the geophysical model components of the Community Climate System Model (CCSM). Currently contains code for CCSM message passing orbital calculations, and system utilities.</td>
</tr>
<tr>
<td>src/ecosysdyn/</td>
<td>Ecosystem dynamics routines (e.g., leaf and stem area index)</td>
</tr>
<tr>
<td>src/main/</td>
<td>Control (driver) routines</td>
</tr>
<tr>
<td>src/mksrdata/</td>
<td>Routines for generating surface datasets</td>
</tr>
<tr>
<td>src/riverroute/</td>
<td>River routing (RTM) routines</td>
</tr>
<tr>
<td>src/utils/</td>
<td>Independent utility routines</td>
</tr>
<tr>
<td>src/utils/esmf/</td>
<td>Earth System Modeling Framework utilities</td>
</tr>
<tr>
<td>src/utils/timing/</td>
<td>General purpose timing library</td>
</tr>
<tr>
<td>bld/</td>
<td>Directory of build, test and run scripts</td>
</tr>
<tr>
<td>bld/offline/</td>
<td>Script to build and execute the model on various platforms</td>
</tr>
<tr>
<td>bld/offline/tests</td>
<td>Perl scripts for model development testing</td>
</tr>
<tr>
<td>doc/</td>
<td>model documentation</td>
</tr>
<tr>
<td>tools/</td>
<td>Directory of tools for input dataset manipulation</td>
</tr>
<tr>
<td>tools/convert_ascii/</td>
<td>Routines for converting user-generated ascii surface dataset files to netCDF format suitable for use by the model (this tool is used independent of running the model)</td>
</tr>
<tr>
<td>tools/cpribindc/</td>
<td>Produces executable that compares model netCDF history files (this tool is used independent of running the model)</td>
</tr>
<tr>
<td>tools/convert_init/</td>
<td>Tool for converting CLM2.0 initial datasets to CLM2.1 form</td>
</tr>
</tbody>
</table>
```

The file CLM2.1_inputdata.tar.gz contains surface and offline atmospheric forcing datasets. This file
must first be uncompressed with the `gunzip` utility and then "untarred" as follows:

```
gunzip -c CLM2.1_inputdata.tar.gz | tar xvf -
```

The above command results in a directory hierarchy containing `inputdata/innd/clm2/` as the root. This directory hierarchy is outlined below.

### Table 2: Input Data Directory Structure

<table>
<thead>
<tr>
<th>Directory Name</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCEPDATA/</td>
<td>One year's worth of atmospheric forcing variables in monthly netCDF format suitable for running the model in offline mode (uncoupled from the atmospheric model)</td>
</tr>
<tr>
<td>inndata_2.1/</td>
<td>Directory hierarchy containing netCDF CLM2.1 initial datasets</td>
</tr>
<tr>
<td>inndata_2.1/cam</td>
<td>Initial datasets for initializing CLM2.1 from a spun-up state when running in cam mode (can also be used when running in offline mode)</td>
</tr>
<tr>
<td>inndata_2.1/ccsm</td>
<td>Initial datasets for initializing CLM2.1 from a spun-up state when running in ccsm mode (can also be used when running in offline mode)</td>
</tr>
<tr>
<td>inndata_2.1/offline</td>
<td>Initial datasets for initializing CLM2.1 from a spun-up state when running in offline mode</td>
</tr>
<tr>
<td>pftdata/</td>
<td>Plant functional type (PFT) physiological constants dataset (ascii format)</td>
</tr>
<tr>
<td>rawdata/</td>
<td>&quot;Raw&quot; (highest provided resolution) datasets (netCDF format) (used by CLM2.1 to generate surface datasets at model resolution)</td>
</tr>
<tr>
<td>rtmdata/</td>
<td>River direction map for RTM in ascii format</td>
</tr>
<tr>
<td>srfdata/</td>
<td>Directory hierarchy containing netCDF CLM2.1 surface datasets</td>
</tr>
<tr>
<td>srfdata/cam</td>
<td>Surface datasets for running CLM2.1 in cam mode (can also be used when running in offline mode)</td>
</tr>
<tr>
<td>srfdata/ccsm</td>
<td>Surface datasets for running CLM2.1 in ccsm mode (can also be used when running in offline mode)</td>
</tr>
<tr>
<td>srfdata/offline</td>
<td>Surface datasets for running CLM2.1 in offline mode</td>
</tr>
</tbody>
</table>

### 3 Creating and Running the Executable

The CLM2.1 model can be built to run in one of three modes. It can run as a stand alone executable where atmospheric forcing data is periodically read in (e.g., using the data in NCEPDATA). This will be referred to as offline mode. It can also be run as part of the Community Atmosphere Model (CAM) where communication between the atmospheric and land models occurs via subroutine calls. This will be referred to as cam mode. Finally, it can be run as a component in a system of geophysical models (CCSM). In this mode, the atmosphere, land, ocean and sea-ice models are run as separate executables that communicate with each other via the CCSM flux coupler. This will be referred to as ccsm mode.

CLM2.1 may be run serially (i.e., on a single processor), in parallel using the Message Passing Interface (MPI) for distributed memory tasks, in parallel using the Open Multi-Processing (OpenMP) directives for shared memory tasks, or finally in parallel using both MPI and OpenMP (hybrid parallelism). For example, the IBM SP consists of distributed memory nodes interconnected by a high performance network connection, and each node contains multiple shared memory processors. When run on the IBM SP, CLM2.1 uses OpenMP directives for parallelism on processors within a shared memory node and MPI routines for
parallelism across distributed memory nodes to take full advantage of the capabilities of the hardware. The configurations supported on each architecture, along with the model run modes, are shown in Table 3.

### Table 3: Supported Architectures

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Run Modes</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM SP</td>
<td>X X X X</td>
<td>X X X X</td>
</tr>
<tr>
<td>SGI</td>
<td>X X X X</td>
<td>X X X X</td>
</tr>
<tr>
<td>Intel</td>
<td>X X X X</td>
<td>X X X X</td>
</tr>
<tr>
<td>Compaq</td>
<td>X X X X</td>
<td>X X X X</td>
</tr>
<tr>
<td>Sparc</td>
<td>X X X</td>
<td>X X X X</td>
</tr>
</tbody>
</table>

The method of building and running CLM2.1 depends on the selected mode as well as the target architecture. A general discussion of the various aspects of building and running CLM2.1 follows.

### 3.1 offline mode: using jobscript.csh

In order to build and run CLM2.1 in offline mode, a sample script, jobscript.csh, and a corresponding Makefile are provided in the `bld/online` directory. In addition, two perl scripts `mkDepends` (used to generate dependencies in a form suitable for inclusion into a Makefile), and `mkSrcfiles` (used to make a list of files containing source code) are also included.

Jobscript.csh creates a model executable at T42 model resolution with RTM river routing activated, determines the necessary input datasets, constructs the input model namelist and runs the model for one day. Users must edit this script appropriately in order to build and run the executable for their particular requirements and in their particular environment. This script is provided only as an example to aid the novice user in getting CLM2.1 up and running as quickly as possible.

The script can be run with minimal user modification, assuming the user resets several environment variables at the top of the script. In particular, the user must set `CSMDATA` to point to the full disk pathname of the root directory containing the un tarred input dataset subdirectories. The user must also set `ROOTDIR` to point to the full disk pathname of the root directory containing the un tarred source code. Finally, the user must set `MODEL_EXEDIR` to point to the directory where the user wants the executable to be built and run.

The script can be divided into five functional sections: 1) specification of script environment variables; 2) creation of the model input namelist; 3) creation of two header files (misc.h and preproc.h) and a directory search path file (Filepath) needed to build the model executable; 4) creation of the model executable; and 5) execution of the model. Jobscript.csh is set up so that the user will normally only have to modify sections 1) and 2) in order to obtain a model executable and associated namelist. Each of these functional sections is discussed in what follows.

#### 3.1.1 Specification of script environment variables

Table 4 lists the user modifiable script environment variables. Some of these variables are used by the Makefile to build the model executable. Although the script provides tentative settings for all these variables, the provided values will generally have to be modified by the user.

### Table 4: User Modifiable Script Variables

<table>
<thead>
<tr>
<th>Script Variable</th>
<th>Description</th>
</tr>
</thead>
</table>

5
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROOTDIR</strong></td>
<td>Full pathname for the root source code directory.</td>
</tr>
<tr>
<td><strong>CSMDATA</strong></td>
<td>Full pathname of root input datasets directory.</td>
</tr>
<tr>
<td><strong>MODEL_EXEDIR</strong></td>
<td>Full pathname for the directory where the model executable will reside.</td>
</tr>
<tr>
<td></td>
<td>Object files will be built in the directory <code>$MODEL_EXEDIR/obj</code>.</td>
</tr>
<tr>
<td><strong>NTHREADS</strong></td>
<td>Number of OpenMP multitasking threads. If set to 1, environment variable SMP is set to FALSE and OpenMP threading is not invoked in the Makefile. NTHREADS should not exceed the number of physical CPUs (ie, processors) on a shared memory machine and should not exceed the number of CPUs in a node on a distributed memory machine.</td>
</tr>
<tr>
<td><strong>NTASKS</strong></td>
<td>Number of MPI tasks. If set to 1, the environment variable SPMD is set to FALSE and distributed memory implementation is not invoked in the Makefile. If NTASKS is greater than 1, distributed memory is enabled.</td>
</tr>
<tr>
<td><strong>LSMLON</strong></td>
<td>Number of model grid longitudes.</td>
</tr>
<tr>
<td><strong>LSMLAT</strong></td>
<td>Number of model grid latitudes.</td>
</tr>
<tr>
<td><strong>LIB_NETCDF</strong></td>
<td>Full pathname of directory containing the netCDF library. The setting depends on user’s target machine.</td>
</tr>
<tr>
<td><strong>INC_NETCDF</strong></td>
<td>Full pathname of directory containing netCDF include files. The setting depends on user's target machine.</td>
</tr>
<tr>
<td><strong>LIB_MPI</strong></td>
<td>Full pathname of directory containing the MPI library. The setting depends on user’s target machine. Only needed if ntasks larger than 1.</td>
</tr>
<tr>
<td><strong>INC_MPI</strong></td>
<td>Full pathname for directory containing the MPI include files. The setting depends on user’s target machine. Only needed if ntasks is larger than 1.</td>
</tr>
<tr>
<td><strong>DEBUG</strong></td>
<td>Turns debugging flags on in Makefile (valid values are TRUE or FALSE).</td>
</tr>
</tbody>
</table>

To obtain a model executable, the environment variables LIB_NETCDF and INC_NETCDF, which provide pathnames to netCDF library and include files must be specified. Furthermore, if the model is to be run under MPI (i.e. NTASKS is greater than 1), then directories containing the MPI library and MPI include files must also be specified as environment variables in the script. (This is not the case on the IBM and the COMPAQ, where the MPI library and include files are obtained directly from the compiler command).

### 3.1.2 Setting the Namelist

Before building and running the model, the user must specify model namelist variables via the namelist, clmexp. A default namelist is generated by jobscript.csh. This namelist results in the generation of a one day model run using the provided datasets. Namelist input is written to the file incstdin and can be divided into several categories: run definitions, datasets, history and restart file settings and land model physics settings. A full discussion of model namelist variables is given in section 4.

### 3.1.3 Creation of header and directory search path files

The user will generally not need to modify the section of jobscript.csh that creates the header and directory search path files. The script creates three files in the directory `$MODEL_EXEDIR/obj`: the header files misc.h and preproc.h and the directory search path file, Filepath. To modify these files, the user should edit the file contents from within the script rather than attempt to edit the files directly, since the script will
overwrite the files upon execution. The use of these files by `gnumake` is discussed in section 3.1.4. Each of these files is summarized below.

The file, `misc.h`, contains resolution- and model-independent C-language pre-processor (cpp) tokens

Table 5: Misc.h CPP tokens

<table>
<thead>
<tr>
<th>misc.h cpp token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPMD</td>
<td>If defined, enables distributed memory, SPMD (single program multiple data), implementation. Automatically defined if environment variable <code>NTASKS</code> &gt; 1. See section 3.1.1.</td>
</tr>
<tr>
<td>PERGRO</td>
<td>If defined, enables modifications that test reasonable perturbation error growth. Only applicable in cam mode (see CAM User’s Guide).</td>
</tr>
</tbody>
</table>

The file `preproc.h` contains resolution-dependent and model-dependent C-language cpp tokens

Table 6: Preproc.h CPP tokens

<table>
<thead>
<tr>
<th>preproc.h cpp token</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFLINE</td>
<td>If defined, offline mode is invoked</td>
</tr>
<tr>
<td>COUP_CSM</td>
<td>If defined, csm mode is invoked</td>
</tr>
<tr>
<td>COUP_CAM</td>
<td>If defined, cam mode is invoked</td>
</tr>
<tr>
<td>LSMLON</td>
<td>Number of model longitudes</td>
</tr>
<tr>
<td>LSMLAT</td>
<td>Number of model latitudes</td>
</tr>
<tr>
<td>RTM</td>
<td>If defined, RTM river routing is invoked</td>
</tr>
</tbody>
</table>

C-preprocessor directives of the form `#include`, `#if defined`, etc., are used in the model source code to enhance code portability and allow for the implementation of distinct blocks of functionality (such as incorporation of different modes) within a single file. Header files, such as `misc.h` and `preproc.h`, are included with `#include` statements within the source code. When `gnumake` is invoked, the C-preprocessor includes or excludes blocks of code depending on which cpp tokens have been defined. C-preprocessor directives are also used to perform textual substitution for resolution-specific parameters in the code. The format of these tokens follows standard cpp protocol in that they are all uppercase versions of the Fortran variables, which they define. Thus, a code statement like

```
parameter(lsmlon = LSMLON); parameter(lsmlat = LSMLAT)
```

will result in the following processed line (for T42 model resolution):

```
parameter(lsmlon=128) ; parameter(lsmlat=64)
```

where LSMLON and LSMLAT are set in `preproc.h` via the jobscript.

Filepath contains a list of directories used by Makefile to resolve the location of source files and to determine dependencies. The search begins in the current directory and proceeds to each directory appearing in Filepath, in the order in which they are specified. All files appearing in these directories will be used unless duplicate files are found. For the case of duplicate files, the first file found will be used by gnumake to create the object file. If user-modified code is introduced, Filepath should contain, as the first entry, the directory containing the user code.

Users can add new search directories by editing `jobscript.csh` under “build Filepath”. The default Filepath directory hierarchy for CLM2.1 is as follows:

7
Table 7: Filepath

<table>
<thead>
<tr>
<th>Source Directories</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MODEL_SRCDIR/main</td>
<td>control routines (history, restart, etc)</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/biogeophys</td>
<td>biogeophysics routines</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/ecosysdyn</td>
<td>ecosystem dynamics routines</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/riverroute</td>
<td>river routing routines</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/camclm_share</td>
<td>code shared between CAM and CLM2</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/csm_share</td>
<td>code shared by all CCSM geophysical model components</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/utils/timing</td>
<td>timing routines</td>
</tr>
<tr>
<td>$MODEL_SRCDIR/mksrdfdata</td>
<td>generation of surface dataset routines</td>
</tr>
</tbody>
</table>

3.1.4 Building the model

The user will generally not need to modify the section of jobscript.csh that builds the model executable. Jobscript.csh invokes `gnumake` to generate the model executable. The file, Makefile, located in the `bld/offline` directory, contains commands used by `gnumake` on each of the supported target architectures. Successful invocation of `gnumake` results in an executable, "clm", along with a log file, "compile_log.clm", documenting the build procedure. Any problems encountered during the build procedure will be documented in this log file. A parallel `gnumake` is achieved in the script by invoking `gnumake` with the `-j` option, which specifies the number of job commands to run in parallel.

3.1.5 Running the executable

The user will generally not need to modify the section of jobscript.csh that runs the model executable. Jobscript.csh will execute the commands required to run the model under the supported target architectures. The model runtime environment is determined by the script environment variables `NTASKS` and `NTHREADS`. If `NTASKS` is greater than 1, OpenMP multitasking will be used for the number of threads specified. If `NTASKS` is greater than 1, the model will be run under MPI for the number of tasks specified. If both are greater than 1 (this should only be used for the IBM(SP) or the COMPaq), then hybrid mode OpenMP/MPI will be used (see section 3.1.1).

Most MPI implementations provide a startup script which accepts the MPI executable as a command line argument. Additional command line arguments allow the user to specify details such as the various machine architectures or number of processes to use for the run. Once MPI has created the specified number of processes, model execution will begin. The collection of active tasks will then compute locally and exchange messages with each other to integrate the model.

Upon successful completion of the model run, several files will be generated in `MODEL_EXEDIR`. These include history, restart, and initialization files (see section 4.3) as well as log files documenting the model execution. These log files will have names of clm.log.YYMMDD-HHMMSS, where YY is the last two digits of the current model year, MM is the month, DD is the day of the month, HH is the hour, MM is the minutes, and SS is the seconds of the start of the model run. Timing files, (e.g. "timing.0"), containing model performance statistics are also generated in the executable directory.

3.2 cam mode

When running the model as part of the CAM executable, CAM build and run scripts must be utilized and the user should refer to the CAM User's Guide for specific details on building and running the CAM executable. We will only discuss some essential points of the CAM build and run scripts here.

The header files, preproc.h and misc.h, as well as the directory search path file, Filepath, are needed for the CAM build procedure in an analogous manner to the CLM2.1 build procedure. The user should
keep in mind that the CLM2.1 directory hierarchy \textbf{MUST appear after} the CAM directory hierarchy in Filepath. CLM2.1 contains several files that have the same name as the corresponding CAM files (e.g. `time_manager.F90`). When running in CAM mode, the corresponding CAM file must be used. The CAM build and run scripts ensure this.

The CLM2.1 namelist, `clmexp`, must also be specified. By default, RTM river routing is not enabled in cam mode (i.e. the `cpp` variable, RTM, is not defined). Furthermore, CLM2.1 does not permit the user to independently set several namelist variables (in particular, those dealing with history file logic and run control logic) when running in cam mode. CLM2.1 will override any user input for these variables with the corresponding values used by the CAM model. This is discussed in more detail in section 4.6.

### 3.3 ccm mode

There will be no separate CCSM release containing CLM2.1.

### 4 Namelist Parameters

CLM2.1 namelist inputs are presented in sections 4.1-4.6 below. In what follows, "mode" has values of "offline", "ccsm", "cam" or "all", corresponding to offline mode, csm mode, cam mode, or all the modes. If a namelist variable setting is listed as \textbf{required}, the value must be set in the namelist in order for the model to execute successfully. If a setting is specified as \textbf{required} and the mode is only given as offline, then that variable must only be specified when running in offline mode. For namelist variable settings not listed as \textbf{required}, the code will provide default settings at initialization. In the following variable descriptions, we refer to examples presented in section 7.

#### 4.1 Specification of run length, run type and initial run date

The following list specifies namelist variables associated with the definition of run case names, run types (restart, initial or branch), model time step, and initial run date.

An initial run starts the model from either initial conditions that are set internally in the code (referred to as arbitrary initial conditions) or from an initial conditions dataset (see namelist variable `FINIDAT`) that enables the model to start from a spun-up state.

A restart run is an exact continuation of a previous simulation from its point of termination. Output from a restart run should be bit-for-bit the same as if the previous simulation had not stopped. Run control variables set in the namelist must be the same as in the run that is being restarted.

A branch run is a new case that uses a restart dataset from a previous simulation to begin the integration. For a branch run, the length of the history interval and the output history fields do not have to be the same as in the control simulation. For example, the branching option can be used to output selected fields more frequently than was the case in the original run or to add new auxiliary history files to the model run.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>type</th>
<th>mode</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASEID</td>
<td>Case name (short identifier for run) (see ex. 1,2,3).</td>
<td>char*32</td>
<td>offline, csm (obtained from atm in cam mode)</td>
<td>required</td>
</tr>
</tbody>
</table>
| CTITLE  | Case title for use within history files (long identifier). | char*80| offline, csm (obtained from atm in cam mode) |`
default: blank

name: NSREST
description: Run type (0 for initial run, 1 for restart, 3 for branch) (see ex. 1, 2, 3).
type: integer
mode: offline, ccs (obtained from atm in cam mode)
default: required

name: DTIME
description: Model time step (seconds) (see ex. 1).
type: integer
mode: offline, must agree with CAM2 in ccs mode, obtained from CAM2 in cam mode
default: required (suggested range: 1200-3600 s)

name: NELAPSE
description: Elapsed run time in model time steps (positive) or days (negative) (see ex. 2).
type: integer
mode: offline (obtained from atm/coupler in cam/ccs mode, respectively)
default: required (if NESTEP not set)

name: NESTEP
description: Ending run time in model time steps (positive) or days (negative) (see ex. 1).
type: integer
mode: offline (obtained from atm/coupler in cam/ccs mode, respectively)
default: required (if NELAPSE not set)

name: START_YMD
description: Start date of run (yyyyymmdd format) (see ex. 1).
type: integer
mode: offline, ccs (obtained from atm in cam mode)
default: required

name: START_TOD
description: Start time of day of run (seconds) (see ex. 1).
type: integer
mode: offline, ccs (obtained from atm in cam mode)
default: 0
4.2 Specification of model input datasets

The following list specifies namelist variables associated with model input datasets.

**name**: FSURDAT
**description**: Full pathname of surface dataset (see ex. 1,2,3).
**type**: char*256
**mode**: all
**default**: blank
**notes**: raw datasets to generate surface dataset provided are in $CSMDATA/rawdata
surface datasets provided with the distribution are in $CSMDATA/srfdata

**name**: FINIDAT
**description**: Full pathname of initial conditions dataset (see ex. 1,2,3).
**type**: char*256
**mode**: all
**default**: blank
**notes**: datasets provided are in $CSMDATA/inidata

**name**: FPFTCON
**description**: Full pathname of plant functional type (PFT) physiological constants dataset
(see ex. 1,2,3).
**type**: char*256
**mode**: all
**default**: required
**notes**: dataset provided is $CSMDATA/pftdata/pft-physiology

**name**: FRIVINP_RTM
**description**: full pathname of RTM input dataset (see ex. 4).
**type**: char*256
**mode**: offline, ccsm
**default**: required if cpp token RTM is defined in preproc.h
**notes**: dataset provided is $CSMDATA/rtmdata/rdirc.05

**name**: NREVSN
**description**: Full pathname of restart file name (only for branch runs) (see ex. 3).
**type**: char*256
**mode**: all
**default**: required (only if branch run, NSREST=3)

**name**: MKSRF_FVEGTYP
**description**: Full pathname of raw vegetation type dataset (see ex. 5).
**type**: char*256
**mode**: all
**default**: required (if FSURDAT is blank)
**notes**: dataset provided is $CSMDATA/rawdata/mksrf_pft.nc
name: MKSRF_FSOITEX
description: Full pathname of raw soil texture dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: dataset provided is $CSMDATA/rawdata/mksrf_soitex.10level.nc

name: MKSRF_FSOICOL
description: Full pathname of raw soil color dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: $CSMDATA/rawdata/mksrf_soicol_clm2.nc

name: MKSRF_FLANWAT
description: Full pathname of raw inland water dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: dataset provided is $CSMDATA/rawdata/mksrf_lanwat.nc

name: MKSRF_FURBAN
description: Full pathname of urban dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: dataset provided is $CSMDATA/rawdata/mksrf_urban.nc

name: MKSRF_FGLACIER
description: Full pathname of glacier dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: dataset provided is $CSMDATA/rawdata/mksrf_glacier.nc

name: MKSRF_FLAI
description: Full pathname of leaf and stem area index, canopy top and bottom height dataset (see ex. 5).
type: char*256
mode: all
default: required (if FSURDAT is blank)
notes: dataset provided is $CSMDATA/rawdata/mksrf_lai.nc

name: MKSRF_OFFLINE_FNAVYORO
description: 20 min navy orography dataset used to generate land mask (see ex. 5).
**MKSRSF_OFFLINE_FGRID**

Description: Dataset specifying land grid and mask at desired resolution (see ex. 6).

Type: `char*256`

Mode: `offline`

Default: `required` (if MKSRF_OFFLINE_FGRID not set and FSURDAT is blank)

Notes: Dataset provided is `$CSMDATA/rawdata/mksrf_navyoro_20min.nc`

**MKSRSF_OFFLINE_EDGEN**

Description: Northern edge of land grid (degrees north) (see ex. 5).

Type: `real`

Mode: `offline`

Default: 90.

**MKSRSF_OFFLINE_EDGEE**

Description: Eastern edge of land grid (degrees east) (see ex. 5).

Type: `real`

Mode: `offline`

Default: 180.

**MKSRSF_OFFLINE_EDGES**

Description: Southern edge of land grid (degrees north) (see ex. 5).

Type: `real`

Mode: `offline`

Default: -90.

**MKSRSF_OFFLINE_EDGEW**

Description: Western edge of land grid (degrees east) (see ex. 5).

Type: `real`

Mode: `offline`

Default: -180.

**OFFLINE_ATMDIR**

Description: Directory containing atmospheric forcing datasets (see ex. 1, 2, 3).

Type: `char*256`

Mode: `offline`

Default: `required`

Notes: Datasets provided are in directory `$CSMDATA/NCEPDATA`

**FSURDAT** specifies a surface dataset containing time-invariant land properties such as plant functional types and soil textures and time-variant properties such as leaf area index. Surface datasets provided with the distribution are contained in the following directories:
If FSURDAT is set to the empty string, a new surface dataset is generated at run time for the specified model resolution and land/ocean mask. The creation of a new surface dataset also requires the specification of the full pathname of the following raw datasets: MKSRF_FVEGTYP, MKSRF_FSOITEX, MKSRF_FSOICOL, MKSRF_FLANWAT, MKSRF_FURBAN, MKSRF_FGLACIER, MKSRF_FLAI. These datasets are only used for the generation of a model surface dataset. They are provided with the distribution and are contained in the input data directory rawdata.

In addition to raw datasets, a land/ocean mask is also required for the creation of a new surface dataset. If the model is run in ccssm or cam mode, this mask is obtained from either the ccssm flux coupler or from the cam atmosphere model at startup. In offline mode, however, the land/ocean mask can either be calculated from a high resolution orography dataset by setting the namelist variable MKSRF_OFFLINE_FNAVYORO or can be read in from an input dataset via the setting of the namelist variable MKSRF_OFFLINE_FGRID.

Subroutines involved in creating a runtime surface dataset reside in the source code directory mksrfdata. In most cases the creation of a surface dataset involves a straightforward interpolation from the raw dataset resolution to the desired model resolution. For soil texture, however, averaging would create new soil types. Consequently, the model determines the dominant soil texture profile per gridcell from the raw resolution to the desired resolution. Once the surface dataset is created, the model reads the necessary surface dataset back in so that the same results will be obtained regardless of whether a run starts from an existing surface dataset or one created at startup.

The input file, FINIDATA, contains values for the time-dependent variables needed to initialize the model from a spun-up state. It is important to note that due to the differences in data structures, CLM2.1 initial datasets are not in the same form as CLM2.0 initial datasets. The directory tools/convert_inic contains routines needed to perform the conversion of CLM2.0 initial datasets to CLM2.1 form. The README file in that directory contains necessary information to build and utilize the conversion tool.

If FINIDATA is set to the empty string, the model is initialized to non spun-up values. The setting of the namelist variable, HIST_CRTINIC (described in the next section), can be used to generate initial files during a model run. Initial datasets provided with the distribution are contained in the directory $CSMDATA/inidata_2.1.

FPFTCON specifies the dataset containing plant functional type physiological constants. The dataset provided with the distribution is $CSMDATA/pftdata/pft-physiology.

When the cpp token RTM is defined, the RTM river routing scheme will be invoked in running the model. In this case, FRIVINP_RTM must be set to a river routing dataset. The dataset provided with the distribution is $CSMDATA/rtmdata/rdirc.05

NREVSN is ignored unless a branch run is specified (i.e., NSREST is set to 3).

In offline mode, time dependent atmospheric forcing data must be read in periodically. The directory containing these files is given by OFFLINE_ATMDIR. Forcing datasets provided with the distribution can be found in $CSMDATA/NCEPDATA. This variable is ignored in cam and ccssm mode.

### 4.3 Specification of history and restart files

The following describes namelist variables associated with history, restart, and initialization files. In what follows, max_tapes denotes the maximum allowable number of different types of history files (tapes) that the model can produce (currently set to 6) and max_flds denotes the maximum number of history fields that may appear on any given history tape (currently set to 1000).

```python
define name: HIST_CRTINIC
```
description: Frequency with which initial datasets will be generated. Valid values are 'MONTHLY','YEARLY' or 'NONE'.

type: char*8
mode: offline, ccs (obtained from atm in cam mode)
default: 'YEARLY'

name: HIST_NHTFRQ(max_tapes)
description: History tape interval(s) (+ for model time steps, - for hours, 0 for monthly ave) (see ex. 4).
type: integer array (1:max_tapes)
mode: offline, ccs (obtained from atm in cam mode)
default: -24

name: HIST_MFILT(max_tapes)
description: Number of time samples per history tape(s) (see ex. 4).
type: integer
mode: offline, ccs (obtained from atm in cam mode)
default: 1

name: HIST_NDENS(max_tapes)
description: Output tape precision(s). Valid values are 1 (double precision) or 2 (single precision).
type: integer
mode: all
default: 2

name: HIST_DOV2XY(max_tapes)
description: Per tape spatial averaging flag. If set to true, produces grid-average history fields on output tape. If set to false, one-dimensional fields are produced (see ex. 4).
type: logical
mode: all
default: .TRUE.

name: HIST_AVGFLAG_PER_TAPE(max_tapes)
description: Per tape time averaging flag. Valid values are 'A' (average over history period), 'I' (instantaneous), 'X' (maximum over history period) or 'M' (minimum over history period).
type: char*1
mode: all
default: blank

name: HIST_TYPE1D_PER_TAPE(max_tapes)
**description:** Per tape one dimensional output type. Only used if one dimensional output is selected for the given tape (via the setting of HIST_DOV2XY). Valid values are 'GRID', 'LAND', 'COLS', 'PFTS'. For example, if one dimensional output is selected for tape 3 and HIST_TYPE1D_PER_TAPE is set to 'COLS', then all the fields will have 1d column output. If the specified one dimensional output type is not defined for a given field, output values will be set to 1.e36 for that field.

**type:** char*4
**mode:** all
**default:** blank

**name:** HIST_EMPTY_HTAPES
**description:** If set to true, all the history tapes are empty by default. Only variables explicitly listed by the user will be output.

**type:** logical
**mode:** all
**default:** .FALSE.

**name:** HIST_FINCL1...HIST_FINCL6(max_ds)
**description:** List of fields to include on the respective history tape. See tables 9-16 for the list of default fields on the primary history tape. Namelist specification can take one of two forms. The user may specify only the name of the field to be included on the history tape (in which case the default time averaging for that field will be used). For example, HIST_FINCL2='TV', will add the field TV to the second history tape with whatever default time averaging was specified for TV. Alternatively, the user may specify the field name, followed by a : followed by the time averaging flag desired (valid flags are 'I' for instantaneous, 'A' for average, 'M' for minimum, and 'X' for maximum). For example, HIST_FINCL2='TV:I' will add the field TV with instantaneous output to the second history tape.

**type:** char*34
**mode:** all
**default:** blank

**name:** HIST_FEXCL1..HIST_FEXCL6(max_fields)
**description:** List of fields to exclude from the respective history tape. The field name must appear in the Master Field List. Currently, there is one default history tape, (the primary tape is monthly averaged). See tables 9-16 for more details.

**type:** char*32
**mode:** all
**default:** blank

**name:** MSS_IRT
**description:** Mass store retention period (days) for output datasets (see ex. 4)

**type:** integer
**mode:** offline, ccsm (obtained from atm in cam mode)
**default:** 0 (i.e., history files will be written to local disk, not the NCAR Mass-Store)
The model writes its own history, restart and initial files. History files are in netCDF file format and contain model data values written at user-specified frequencies during a model run. Each field has a default time averaging flag determining how that field will be accumulated in time over a given history interval. The choices are to record averaged, instantaneous, maximum, or minimum values. The user may overwrite this default setting via the namelist variable HIST_FINCLX where X can equal 1 to 6. If the user wishes to see a field written at more than one output frequency (e.g., daily and hourly), additional history files must be declared containing that field. By default, CLM2.1 produces a monthly averaged primary history file and allows the user to define up to five auxiliary history files. All files contain grid averaged data unless the namelist variable HIST_DOV2XY is set to false for a given file. Primary history files contain the string ‘h0’, whereas auxiliary history files contain the string ‘h1’, ‘h2’, ‘h3’, ‘h4’ and ‘h5’. Monthly averaged history files may be produced for history tape “t” by setting HIST_NHTFRQ(t) to zero. By default, all time averaged fields on tape t will be output over the period beginning from the first timestep of the current month up to and including the last timestep of that month. Each monthly history file will contain exactly one time slice of data, regardless of the value of HIST_MFILT(t).

The model will also periodically create initial netCDF datasets containing only one dimensional instantaneous values of initial data fields. The model produces initial datasets either yearly, monthly, or not at all depending on the setting of the namelist variable HIST_CRNINC (the default setting is “YEARLY”). These datasets can be utilized as “spun-up” initial conditions.

Restart files are in binary format and can be used only for restart or branch runs from previous model simulations. Whenever a restart file is written, a corresponding local disk restart pointer file is overwritten. The restart pointer file contains the name of the latest model restart file. By default, the restart pointer file is placed in the user’s home directory under the name, lnC.CASEID.rpointer. The user may modify the full pathname of the restart pointer file via the setting of the namelist variable RPNTPATH.

The following table specifies the naming convention used for output files. In this table the string yyyy refers to the model year, mm refers to the model month, dd refers to the model day and sssss corresponds to seconds into the model day. Note that for non-monthly history files, yyyy-mm-dd-ssss corresponds to the first timestamp of data on the file. CASEID is the case identifier set via the namelist input.

CASEID.clm2.r. yyyy-mm-dd-ssss   restart files
CASEID.clm2.i. yyyy-mm-dd-ssss.nc   initial files
CASEID.clm2.h i012345 . yyyy-mm.nc | monthly average history files
CASEID.clm2.h i012345 . yyyy-mm-dd-ssss.nc | non-monthly history files

History, restart and initialization files can be archived on the NCAR Mass Storage System (MSS) if the namelist variable MSS_IRT is set to a value greater than zero. History, restart and initial files are archived as follows (where USERNAME is the upper-case equivalent of the user’s login name, i.e., the user’s root directory on the MSS):
history files /USERNAME/csm/CASEID/lnd/hist
restart files /USERNAME/csm/CASEID/lnd/rest
initial files /USERNAME/csm/CASEID/lnd/init

4.4 Specification of input physics variables

name: CONCHK
description: Turns on error checks for energy and water conservation.
type: logical
mode: all
default: .TRUE.

name: IRAD
description: Frequency of solar radiation calculations (+ for model time steps, - for hours).
type: integer
mode: offline, must be consistent with CAM2 in csm mode obtained from atm in cam mode
default: -1

ame: CSM_DOFLXAVE
description: If set to true, flux averaging is performed over the duration set in IRAD.
type: logical
mode: csm (must agree with CAM2 setting in atm.setup.csh)
default: .TRUE.

name: WRTDIA
description: If true, global average 2-m temperature written to standard out (ascii log file of the run) (see ex. 4).
type: logical
mode: all
default: .FALSE.

4.5 Specification of RTM River routing

name: RTM_NSTEPS
description: Number of time steps over which RTM output will be averaged.
type: integer
mode: all
default: number of timesteps in 3 hours
4.6 Specification of cam mode namelist

When running in cam mode, certain CLM2.1 namelist variables cannot be set independently. In particular, any user specification for the namelist variables, CASEID, CTITLE, IRAD, NSREST, HIST_CRTINIC, HIST_NHTFRQ(1), HIST_MFILT(1) and MSS_IRT will be overwritten by values obtained from CAM2 at startup. All other namelist settings may be set independently by the user.

The following table specifies the namelist variables that are overwritten with values obtained from cam and lists the associated CAM2 namelist variable and its default value.

<table>
<thead>
<tr>
<th>CLM Namelist</th>
<th>CAM namelist</th>
<th>CAM default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASEID</td>
<td>CASEID</td>
<td>required</td>
</tr>
<tr>
<td>CTITLE</td>
<td>CTITLE</td>
<td>blank</td>
</tr>
<tr>
<td>NSREST</td>
<td>NSREST</td>
<td>0</td>
</tr>
<tr>
<td>IRAD</td>
<td>IRADSW</td>
<td>-1</td>
</tr>
<tr>
<td>HIST_CRTINIC</td>
<td>INITHIST</td>
<td>&quot;MONTHLY&quot;</td>
</tr>
<tr>
<td>HIST_NHTFRQ(1)</td>
<td>NHTFRQ(1)</td>
<td>0</td>
</tr>
<tr>
<td>HIST_MFILT(1)</td>
<td>MFILT(1)</td>
<td>1</td>
</tr>
<tr>
<td>MSS_IRT</td>
<td>MSS_IRT</td>
<td>365</td>
</tr>
</tbody>
</table>

The minimum set of model input namelist variables that must be given values by the user at run time depends on if a pre-existing surface dataset already exists and if RTM river routing is enabled. At a minimum, only the namelist variables FPFTCON and FSURDAT must be given values if a model surface dataset exists and if the cpp variable RTM is not defined. If a monthly surface dataset does not exist, then MKSRF_FVEGTYP, MKSRF_FSOITEX, MKSRF_FSOICOL, MKSRF_FLANWAT, MKSRF_FGLACIER, MKSRF_FURBAN, and MKSRF_FLAI must be given values in order to generate one at runtime. If RTM is invoked, then FRIVINP_RTM must also be specified. Finally, the namelist variable FINIDAT must be set if a pre-existing initial dataset is to be utilized.

5 CLM2.1 Data Structures

The hierarchical subgrid data structures contained in CLM2.1 are implemented in the code through the modules clntype.F90, clm_mapping.F90 and clmpoint.F90 (all in the /src/main subdirectory). The new code makes extensive use of the Fortran 90 implementation of the derived data type. This permits the user to define new data types that can consist of multiple standard data types (integers, doubles, strings) as well as other derived data types.

The entire definition of the CLM2.1 subgrid hierarchy is contained in module clntype.F90. The module, clntype.F90, is organized such that derived types which are members of other derived types are defined first (a Fortran 90 compiler requirement). In particular, the energy and mass conservation data types are defined first, followed by data types constituting the pft level, column level, landunit level, gridcell level and the model domain level. Finally, the hierarchical organization of these types is defined, starting with the model domain level, which consists in part of a pointer to an array of gridcells, each of which consists in part of a pointer to an array of landunits, each of which has a pointer to an array of columns, which each have a pointer to an array of pfts. The last section of clntype.F90 includes additional data structures that map the hierarchical organization to simple 1d vectors at each level for history, restart and initial file output.

The basic functions of subroutine clm_map() in clm_mapping.F90 are memory allocation for the subgrid hierarchy and initialization of areas and weights associated with each subgrid component. Use is made of input gridded datasets defining the spatial distribution of pfts and other surface types (glacier, lake, etc.). This is the primary routine that needs to be modified in order to accommodate different subgrid representations using the data structures defined in clntype.F90.
Module `clmpoint.F90` creates arrays of one-dimensional real and integer pointers for various `clntype` derived type components. These arrays are integral to the CLM2.1 history file handler logic as well as the restart and initial file generation. The exact usage of these arrays in the model's history file logic is discussed in more detail in section 6.

Finally, CLM2.1 includes the module, `pft2columnMod.F90`, which contains methods for averaging fluxes and states between levels in the hierarchy (currently pft to column in the current implementation). It is intended that similar routines for averaging from column to landunit and from landunit to gridcell will be added later.

6 History File Fields

The following sections discuss both the fields that may currently be output to CLM2.1 history tapes as well as code modifications that the user must make to add new user-defined fields to the history tapes. Understanding of the CLM2.1 data structures is essential before the user attempts to modify code and/or add new history output fields to the model.

6.1 Model history fields

Tables 9-16 list the fields that currently may be output to a CLM2.1 history tape. By default, these fields are also on the primary history tape. The dimensions of each field may include `time` (days since the beginning of the simulation), `levsoi` (number of soil layers, `levsoi = 10`) and `lat` and `lon` (number of latitude and longitude points, e.g., `lat=64, lon=128` for a T42 simulation), for grid averaged two dimensional output and `gridcell`, `landunit`, `column` or `pft` for one dimensional output. Note that the 1d dimension type appearing in the dimensions entry specifies only the default 1d output type. For example, 'TSA' will be output by default in column 1d output. However, that default type may be changed for a given history tape via the setting of the namelist variable `HIST_TYPE1D_PERTAPE`. The Level column can contain either SL or ML as an entry, denoting a single-level or multi-level field, respectively. Finally, unless explicitly specified in the description, all fields are time averaged over the requested history interval.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSA</td>
<td>2 m air temperature</td>
<td>K</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TV</td>
<td>vegetation temperature</td>
<td>K</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TG</td>
<td>ground temperature</td>
<td>K</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TSOI</td>
<td>soil temperature</td>
<td>K</td>
<td>column</td>
<td>ML</td>
<td>lakes excluded</td>
</tr>
<tr>
<td>TLAKE</td>
<td>lake temperature</td>
<td>K</td>
<td>column</td>
<td>ML</td>
<td>nonlakes excluded</td>
</tr>
<tr>
<td>TSNOW</td>
<td>snow temperature</td>
<td>K</td>
<td>column</td>
<td>SL</td>
<td>lakes excluded</td>
</tr>
</tbody>
</table>

Table 10: Master Field List -Surface Radiation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSA</td>
<td>absorbed solar radiation</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FSR</td>
<td>reflected solar radiation</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>NDVI</td>
<td>surface normalized difference vegetation index</td>
<td>unitless</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FIRA</td>
<td>net infrared (longwave) radiation</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FIRE</td>
<td>emitted infrared (longwave) radiation</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>
Table 11: Master Field List - Surface Energy Fluxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCTR</td>
<td>canopy transpiration</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global (set to 0 over lakes)</td>
</tr>
<tr>
<td>FCEV</td>
<td>canopy (intercepted) evaporation</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global (set to 0 over lakes)</td>
</tr>
<tr>
<td>FGEV</td>
<td>ground evaporation</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FSH</td>
<td>sensible heat</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FGR</td>
<td>heat flux into snow/soil (includes snow melt)</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FSM</td>
<td>snow melt heat flux</td>
<td>watt/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TAUX</td>
<td>zonal surface stress</td>
<td>kg/m/s²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TAUY</td>
<td>meridional surface stress</td>
<td>kg/m/s²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>

Table 12: Master Field List - Vegetation Phenology

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELAI</td>
<td>exposed one-sided leaf area index</td>
<td>m²/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>ESAI</td>
<td>exposed one-sided stem area index</td>
<td>m²/m²</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>

Table 13: Master Field List - Canopy Physiology

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSUN</td>
<td>sunlit leaf stomatal resistance (minimum over time interval)</td>
<td>s/m</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>RSSHA</td>
<td>shaded leaf stomatal resistance (minimum over time interval)</td>
<td>s/m</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>BTRAN</td>
<td>transpiration beta factor (soil moisture limitation)</td>
<td>unitless</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>FPSN</td>
<td>photosynthesis</td>
<td>unitless</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
</tbody>
</table>

Table 14: Master Field List - Hydrology

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2OSOI</td>
<td>volumetric soil water</td>
<td>mm³/mm³</td>
<td>column</td>
<td>ML</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>H2OSNO</td>
<td>snow depth (liquid water equivalent)</td>
<td>mm</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>H2OCAN</td>
<td>intercepted water</td>
<td>mm</td>
<td>column</td>
<td>SL</td>
<td>global (set to 0 over lakes)</td>
</tr>
<tr>
<td>SOILLIQ</td>
<td>soil liquid water</td>
<td>kg/m²</td>
<td>column</td>
<td>ML</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>SOILICE</td>
<td>soil ice</td>
<td>kg/m²</td>
<td>column</td>
<td>ML</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>SNOWLIQ</td>
<td>snow liquid water</td>
<td>kg/m²</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>SNOWICE</td>
<td>snow ice</td>
<td>kg/m²</td>
<td>column</td>
<td>SL</td>
<td>excludes lakes</td>
</tr>
<tr>
<td>SNOWDP</td>
<td>snow height</td>
<td>m</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>SNOWAGE</td>
<td>snow age</td>
<td>unitless</td>
<td>column</td>
<td>SL</td>
<td>global (set to 0 over lakes)</td>
</tr>
<tr>
<td>QINFL</td>
<td>infiltration</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>
Table 14: Master Field List - Hydrology

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>QOVER</td>
<td>surface runoff</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QRWGL</td>
<td>surface runoff at glaciers, wetlands, lakes</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QDRAI</td>
<td>sub-surface drainage</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QINTR</td>
<td>interception</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QHDRP</td>
<td>throughfall</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QMELT</td>
<td>snow melt</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QSOIL</td>
<td>ground evaporation</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QVEGE</td>
<td>canopy (intercepted) evaporation</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QVEGT</td>
<td>canopy transpiration</td>
<td>mm/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QCHO CNR</td>
<td>RTM river discharge into ocean</td>
<td>m3/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QCHANR</td>
<td>RTM river flow (maximum subgrid flow)</td>
<td>m3/s</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>

Note that for snow related fields (e.g. SNOWLIQ), horizontal averaging is done only using columns that have snow. In this horizontal averaging lake subgrid points are excluded. Furthermore, for snow related fields, vertical averaging is done by summing only over valid snow layers.

Table 15: Master Field List - Water and Energy Balance Checks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRSOI</td>
<td>soil/lake energy conservation error</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>ERRSEB</td>
<td>surface energy conservation error</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>ERRSOL</td>
<td>solar radiation conservation error</td>
<td>watt/m2</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>ERRH2O</td>
<td>total water conservation error</td>
<td>mm</td>
<td>column</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>

Table 16: Master Field List - Atmospheric Forcing

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Units</th>
<th>1d Output</th>
<th>Level</th>
<th>Spatial Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAIN</td>
<td>rain</td>
<td>mm/s</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>SNOW</td>
<td>snow</td>
<td>mm/s</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>TBOT</td>
<td>atmospheric air temperature</td>
<td>K</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>WIND</td>
<td>atmospheric wind velocity magnitude</td>
<td>m/s</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>THBOT</td>
<td>atmospheric air potential temperature</td>
<td>K</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>QBOT</td>
<td>atmospheric specific humidity</td>
<td>kg/kg</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>ZBOT</td>
<td>atmospheric reference height</td>
<td>m</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FLDS</td>
<td>incident longwave radiation</td>
<td>watt/m2</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
<tr>
<td>FSDS</td>
<td>incident solar radiation</td>
<td>watt/m2</td>
<td>gridcell</td>
<td>SL</td>
<td>global</td>
</tr>
</tbody>
</table>
6.2 Adding new history fields

Model history output may appear in two-dimensional grid form or one-dimensional subgrid form (depending on the value of the namelist variable HIST_DO_V2XY). One-dimensional subgrid output may in turn appear in gridcell, landunit, column or pft form. History file output is controlled by the files histFileMod.F90 and clmpoint.F90 (in directory main/). The user must modify these two files accordingly in order to add new user-defined history fields to the history tapes. It is assumed in the following that the user is completely familiar with Fortran 90 pointer concepts and syntax.

Module clmpoint.F90 creates arrays of one-dimensional real pointers for selected clmtype components. Each of these arrays has one of the following forms:

real pointers to single level subgrid components:
  clmpointer(derived_type_name)%val(one_dimensional_index)%rp

real pointers to multi level subgrid components:
  clmpointer(derived_type_name)%val(one_dimensional_index)%rap(:)

The "derived_type_name" specifies the particular clmtype derived type component that will be pointed to. The "one_dimensional_index" spans the number of model gridcells, landunits, columns or pfts that is appropriate for the "derived_type_name". At the top of clmpoint.F90 a default list of "derived_type_name" parameter settings is provided. For example, ip_pes_t_ref2m corresponds to the "derived_type_name" for the array containing pointers to all pft subgrid level t_ref2m values and one_dimensional_index can have any value from 1 to the number of model subgrid level pfts. Note that the default list of parameter names is much larger than those that are actually used in clmpoint.F90 to set up the pointer arrays.

As a simple illustration, the following example code summarizes how pointer arrays are created and used in clmpoint.F90 (refer to files clmtype.F90 and clm_mapping.F90 for details on clmtype components).

```fortran
! Define clmpointer data structure

type clm_pointer
  type (value_pointer), dimension(:), pointer :: val
end type clm_pointer

integer, parameter :: max_mapflds = 500

! Determine 'derived_type_name' for t_ref2m at the pft level

integer, parameter :: ip_pes_t_ref2m = 1

! Determine beginning and ending pft 1d indices for the current task (see clmtype.F90)
! pft 1d indices correspond to a one-dimensional vector representation of the model subgrid level pft components

begp = pfts1d%beg
endp = pfts1d%end

! Allocate memory for t_ref2m pointer array
allocate (clmpointer(ip_pes_t_ref2m)%val(begp:endp))

! Set up clmpointer array for t_ref2m
! 'g' is a pointer to a model gridcell component
```
! 'l' is a pointer to a landunit component within gridcell 'g'
! 'c' is a pointer to a column component within landunit 'l'
! 'p' is a pointer to a pft component within column 'c'
! 'p%pps' is the physical state component of pft 'p'
! 'p%pps%index1d' is an index into a one dimensional vector
! representation of model subgrid level pft components

! Loop over gridcells
do gi = 1,clm%mps%ngridcells
  g => clm%g(gi)

  ! Loop over all landunits in the given gridcell
do li = 1,g%gps%nlandunits
    l => g%l(li)

    ! Loop over all columns in the given landunit
do ci = 1,l%lps%ncolumns
      c => l%c(ci)

      ! Loop over all pfts in the given column
      do pi = 1,c%cps%npfts
        p => c%p(pi)
pindex = p%pps%index1d

        ! Set up pft level pointer array to t_ref2m
        clmpointer(ip_pes_t_ref2m)%val(pindex)%rp => p%pes%t_ref2m
      end do ! end of PFTs loop
    end do ! end of columns loop
  end do ! end of landunits loop
end do ! end of gridcells loop

In the above example, pindex spans the total number of model subgrid level pfts for a given task (if MPI is used) or the total number of model subgrid level pfts (if MPI is not used). Setting up the above pointer array is analogous to gathering all the 2m reference temperatures from all the model subgrid level pfts into a single one-dimensional array. History file output is done ONLY via these one dimensional pointer arrays set up in clmpoint.F90.

Module histFileMod.F90 contains routines that create and write model history files and that update the model history buffer during the course of the model simulation. The initialization of history fields is done in subroutine masterlist_build(), whereas the determination of the history fields which are active by default on the various history tapes is performed in subroutine masterlist_change_active().

In masterlist_build(), two entries must be specified for each field: an array hpindices followed by a call to subroutine masterlist_addfits(). The array, hpindices, contains four entries: hpindices(1) corresponds to the gridcell level derived_type_name for the requested field, hpindices(2) is the landunit subgrid level derived_type_name for the field, hpindices(3) is the column subgrid level derived_type_name and hpindices(4) is the pft subgrid level derived_type_name. If the value of an hpindices element is set to -1, one-dimensional output is not permitted for that subgrid type for the given field. Requesting such output will result in model termination. If the value of an hpindices element is set to not_valid, then requesting one-dimensional output for that subgrid type will result in a value of 1.e36 appearing for all the output field values.

As an example, hpindices is set as follows for history field SNOWAGE:
Consequently, the user may not request one-dimensional gridcell or landunit output for SNOWAGE. If the user requests one-dimensional pft level output for SNOWAGE via the following namelist settings,

```
HIST_FINC1L2 = 'SNOWAGE'
HIST_TYPE1D_PERTAPE(2) = 'PFTS'
```

each output pft value for SNOWAGE will be 1.e36. Since SNOWAGE is computed in the model as a column property, it makes sense to set its pft derived_type_name to not_valid.

As a second example, the history field, TSA (2m reference temperature) has hpindices set as follows:

```
hpindices = (/-1, -1, ic_ces_pes_a_t_ref2m, ip_pes_t_ref2m/)
```

This implies that one dimensional history output for TSA can occur either on columns or pfts. Gridcell and landunit one dimensional output for this field is not permitted due to the value of -1. Furthermore, one dimensional column output will use the array

```
clmpointer(ic_ces_pes_a_t_ref2m)%val(:)%rp.
```

whereas one dimensional pft output will use the array

```
clmpointer(ip_pes_t_ref2m)%val(:)%rp.
```

Following the setting of hpindices, a call to subroutine `masterlist_addfld()` must be made specifying required initialization information for a given history field. For example, the following call is made to initialize the 2m reference temperature:

```
call masterlist_addfld (fname='TSA', type1d='column', units='K', numlev=1, &
   avgflag='A', long_name='2m air temperature', hpindices=hpindices)
```

The arguments to the above call are as follows:

```
fname : name of history field
type1d : default one dimensional output type
   (valid values are 'gridcell', 'landunit', 'column' or 'pft')
units : field units
numlev : number of vertical values
avgflag : default time averaging flag
   (valid values are 'A'(average), 'I'(instantaneous),
    'M'(minimum) or 'X'(maximum))
long_name : descriptive name of history field
hpindices : array of one dimensional output parameter names.
```

It is important to note that if the namelist variable `HIST_DO2XY` is set to true, history output will appear in two dimensional grid form. Two-dimensional grid output will be obtained by calculating grid cell averages from the "type1d" subgrid components.

The following lists steps that must be taken by the user to add a new history field to the model. Currently, the upward longwave radiation above the canopy is not output to the history file. We assume that the user wants to add this field to the history output at either the column or pft subgrid level. The following steps assume that a clmpointer array does not exist for the desired history variable. Steps 1 and 2 below discuss how such a pointer array is created. If a clmpointer array already exists for the needed history variable then only steps 4 and 5 are needed. Step 3 is needed only if 1d output is requested at a subgrid level where spatial averaging is required and that spatial averaging is not currently in the code.
1. Currently, parameter values for the `derived_type_names` of column and pft level upward longwave radiation are already contained at the top of `clmpoint.F90` (`ip_pef_ulrad` and `ic_cef_pef_a_ulrad`). If these parameters did not already exist, they would have to be added to the list of existing parameter values.

2. The following code fragments should be added to subroutine `clmpoint_init()` in module `clmpoint.F90`.

```fortran
! Add code to allocate memory for pft level pointer array
! for upward longwave radiation
allocate (clmpointer(ip_pef_ulrad)%val(begp:endp))

! Add code to allocate memory for column level pointer array
! for upward longwave radiation
allocate (clmpointer(ic_cef_pef_a_ulrad)%val(begc:endc))

! Add code to set up column and pft level clmpointer arrays for
! upward longwave history output
! 'g' is a pointer to a model gridcell component
! 'l' is a pointer to a landunit component within gridcell 'g'
! 'c' is a pointer to a column component within landunit 'l'
! 'c%cps' is the physical state component of column 'c'
! 'c%cps%index1d' is an index into a one dimensional vector
! representation of model subgrid level column components
! 'p' is a pointer to a pft component within column 'c'
! 'p%pps' is the physical state component of pft 'p'
! 'p%pes' is the energy state component of pft 'p'
! 'p%pps%index1d' is an index into a one dimensional vector
! representation of model subgrid level pft components

! Loop over gridcells
do gi = 1,clm%mps%ngridcells
  g => clm%g(gi)

  ! Loop over all landunits in the given gridcell
  do li = 1,g%gps%nlandunits
    l => g%l(li)

    ! Loop over all columns in the given landunit
    do ci = 1,l%lps%ncolumns
      c => l%c(ci)
      cindex = c%cps%index1d

      ! Set up column level pointer array
      clmpointer(ic_cef_pef_a_ulrad)%val(cindex)%rp => c%cef%pef_a%ulrad

      ! Loop over all pfts in the given column
      do pi = 1,c%cps%npfts
        p => c%p(pi)
        pindex = p%pps%index1d
```

26
Set up pft level pointer array
clmpointer(ip_pef_ulrad)%val(pindex)%rp => p%pef%ulrad

end do ! end of PFTs loop
end do ! end of columns loop
end do ! end of landunits loop
end do ! end of gridcells loop

3. Currently, the model calculates the upward longwave radiation at both the pft and column subgrid level. Pft level values are obtained in module CanopyFluxesMod.F90 whereas column level values are calculated in module pft2columnMod.F90.

4. The following statements should be added to subroutine masterlist_build() (in module histFileMod.F90):

hpindices = (/ -1, -1, ic_cef_pef_a_ulrad, ip_pef_ulrad /)
call masterlist_addfld (fname='ULRAD', type1d='pft', units='W/m2',
  numlev=1, avgflag='A', long_name='upward longwave radiation above canopy',
  hpindices=hpindices)

Note that setting type1d to 'pft' above will result in default 1d output on the pft subgrid level for the history field 'ULRAD'.

5. The following statement should be added to subroutine masterlist_change_active() (in module histFileMod.F90) if the user wants this field on the primary tape by default:

call masterlist_make_active (name='ULRAD', tape_index=1)

7 Offline Mode Namelist Examples

The following examples illustrate different namelist options that can be used to run CLM2.1 in offline mode.

7.1 Example 1: Offline initial run, one day, global

When the model is run in offline mode using a pre-existing surface dataset, the minimum namelist parameters are: CASEID, NSREST, NESTEP or NELAPSE, FSURDAT, FPFTCON, OFFLINE_ATMDIR, START_YMD, and DTIME. If FSURDAT is blank, a surface dataset will be generated at run time and additional variables need to be specified (see section 4.2 and Examples 5 and 6). All other namelist parameters will be set to default values. The following gives an example of a simple namelist.

```clmexp
&clmexp
  CASEID = 'test01'
  NSREST = 0
  NESTEP = -1
  FSURDAT = '/$CSMDATA/srfdata/cam/clms_64x128_T42_c020514.nc'
  FINIDAT = '',
  FPFTCON = '/$CSMDATA/pftdata/pft-physiology'
  FRIVINP_RTM = '/$CSMDATA/rtmdata/rdirc.05'
  OFFLINE_ATMDIR = '/$CSMDATA/NCEPDATA'
  START_YMD = 19971231
  DTIME = 1800
```

27
CASEID = 'test01'
Case identifier which distinguishes this particular simulation from another. The string in CASEID shows up in the names of history, restart, and initial files, in the restart pointer file name (see Example 2) and in the Mass Store pathname where history and restart files are placed if the Mass Store is used. In a branch run, the user must specify a new CASEID.

NSREST = 0
Requests an initial run, as opposed to a restart or a branch run. An initial run does not require the use of an initial input data file (FINIDAT). If none is provided, the model uses non spun-up initialization provided in the code (see src/main/IniTimeVar.F90).

NESTEP = -1
Specifies the run’s ending time to be at the end of day 1. Since NESTEP overrides any value given to NELAPSE, NELAPSE has been omitted in this example.

FSURDAT = '$CSMDATA/srfdata/cam/clms_64x128_T42_c020514.nc'
Specifies the name of the surface data input file. This T42 surface dataset can be used both in cam and offline mode. The model resolution, (i.e. parameters lsmlon and lsmlat) must be compatible with the resolution of FSURDAT. If the filename appeared without a path specifying its exact location, the file would be expected in the executable directory, defined by the environment variable $MODEL_EXEDIR.

FINIDAT = ' ' Specifies that model-specified initial values be used.

FPFTCON = '$CSMDATA/pftdata/pft-physiology'
Specifies a file with PFT (Plant Functional Type) information.

FRIVINP_RTM = '$CSMDATA/rtmddata/rdirc.05'
Specifies the input file required for the operation of RTM (the River Transport Model of Branstetter et al). By default, RTM will operate at half degree horizontal resolution and will be invoked every 3 hours, where the RTM input fluxes are averaged over the 3 hour period. If the user wants the RTM scheme to be invoked every timestep, RTM_NSTEPS should be set to 1. If the user wants the RTM scheme to operate at a different frequency than once every 3 hours, RTM_NSTEPS should be set to the desired value. Use of RTM is activated in the jobscript.csh with the C pre-processor (cpp) directive #define RTM in the header file preproc.h (see 3.1.3).

OFFLINE_ATMDIR = '$CSMDATA/NCEPDATA'
Specifies the location of the atmospheric driver data set. Such a data set is required for the model to run in offline mode.

START_YMD = 19971231
Specifies the base date of the simulation and must be compatible with the atmospheric input data. For example, START_YMD = 19971231 will use the atmospheric input file 1997-12.nc. In a restart or branch run, START_YMD need not be changed, as long as it refers to a date earlier than the date of restart or branch.

DTIME = 1800
Specifies the simulation’s timestep in seconds. In offline mode, the model can handle a timestep of up to 3600 seconds.

HIST_NHTFRQ(1) = -24
Primary history files and restart files will be produced in the executable directory and will be written every 24 hours.
7.2 Example 2: Restart run

The following namelist generates a restart run starting from the last file generated by Example 1.

```clmexp
CASEID        = 'test01',
NSREST        = 1,
NELAPSE       = -1,
FSURDAT       = '%$CSMDATA/srfdata/cam/clms_64x128_T42_c020514.nc',
FINIDAT       = '',
FPFTCON       = '%$CSMDATA/pftdata/pft-physiology',
FRIVINP_RTM   = '%$CSMDATA/rtmdata/rdirc.05',
OFFLINE_ATMDIR= '%$CSMDATA/NCEPDATA',
START_YMD     = 19971231,
DTIME         = 1800,
HIST_NHTFRQ(1)= -24,
/
```

**NSREST = 1**

Requests a restart run. A restart run finds the name of the appropriate restart file automatically by reading the file, `CASEID.rpointer`. In this example, the pointer file will be `ind.test01.rpointer`. Restart runs are meant to be 'seamless,' producing the same output as runs which were not restarted.

**NELAPSE = -1**

Specifies the run's ending time to be one day after the point of restart. This is equivalent to entering `NELAPSE = -2` instead, since the previous run stopped at the end of day 1.

All other namelist variables remain the same to ensure a 'seamless' restart (for information, see example 1). Also, for a seamless restart, the user should generally execute the code with the same executable used in the initial run (i.e., without compiling the code again).

7.3 Example 3: Branch run

The following namelist generates a branch run starting from restart files generated by Example 1. The user may branch a run with the same executable used in the initial run (i.e., without recompiling the code) unless branching is used to test changes in the code (for debugging or sensitivity purposes).

```clmexp
CASEID        = 'branch_run',
NSREST        = 3,
NELAPSE       = -1,
FSURDAT       = '%$CSMDATA/srfdata/cam/clms_64x128_T42_c020514.nc',
FINIDAT       = '',
FPFTCON       = '%$CSMDATA/pftdata/pft-physiology',
FRIVINP_RTM   = '%$CSMDATA/rtmdata/rdirc.05',
OFFLINE_ATMDIR= '%$CSMDATA/NCEPDATA',
NREVSN        = 'test01.clm2.r.1998-01-01-00000',
HIST_FINCL2   = 'TV:I',
HIST_NHTFRQ   = -3,5,
HIST_MFILT    = 2,3,
START_YMD     = 19971231,
DTIME         = 1800,
/
```

29
See Example 1 for explanations of namelist variables which remain unchanged.

**NSREST** = 3
Requests a branch run.

**NELAPSE** = -1
Specifies the run's ending time to be one day after the point of branching.

**NRREVSN** = 'test.clm2.r.1998-01-01-00000'
Supplies the name of the restart file which will initialize this run. (Note this file can be produced by running example 1 above).

**HIST_FINCL2** = 'TV:I'
Add an auxiliary history file with the field “TV” that is output instantaneously.

**HIST_NHTFRQ** = -3.5
Changes the frequency of primary history history writes to every 3 hours. The write frequency of the auxiliary file is every 5 time steps. This is an example of a change which a user may wish to test in a branch run.

**HIST_MFILT** = 2,3
The primary history file will have 2 time samples on every tape. The auxiliary history file will have 3 time samples on every tape.

### 7.4 Example 4: Auxiliary history files

This example covers the addition of an auxiliary history file, the removal of a field from the primary history file and the change of field type in a history file. A variety of other namelist options are also illustrated.

```
&CLMEXP
  CASEID = 'rtm_run'
  NSREST = 0
  NESTEP = -31
  START_YMD = 19980101
  DTIME = 1800
  FSURDAT = '$CSMDATA/srfdata/cam/clms_64x128_T42_c020514.nc'
  FINIDAT = '',
  FPFTCON = '$CSMDATA/pftdata/pft-physiology'
  FRTINP_RTM = '$CSMDATA/rtmdata/rdirc.05'
  OFFLINE_ATMDIR = '$CSMDATA/NCEPDATA'
  HIST_DOV2XY = .true.,.false.
  HIST_NHTFRQ = -24,-12
  HIST_MFILT = 4,2
  HIST_FINCL2 = 'TV','TG:I'
  HIST_FEXCL1 = 'TSNOW'
  MSS_IRT = 365
  WRTDIA = .true.
/
```

For namelist variables which are repeated, refer to Examples 1, 2, and 3.

**HIST_DOV2XY** = .true.,.false.
History output will appear in gridded two-dimensional format (rather than one-dimensional subgrid format) for the primary file and in one-dimensional subgrid format for the auxiliary file.
HIST_NHTFRQ = -24,-12
History output will be directed to the primary history file every 24 model hours and to the auxiliary history file every 12 hours.

HIST_MFILT = 4,2
Each primary history file will contain 4 time slices of output, while each auxiliary history file will contain 2 time slice of output.

HIST_FINC12 = 'TV', 'TG:1'
Specifies the two fields to be added to the auxiliary history output. The first field, 'TV', will have the default time averaging done, whereas the second field, 'TG' will have instantaneous output.

HIST_FEXCL1 = 'TSNOW'
The field 'TSNOW' will be excluded from the primary tape.

WRTDIA = .true.
A global average of land surface air temperature as diagnostic will appear in the standard output file of the simulation.

MSS_IRT = 365
Output files will be archived to the NCAR Mass Storage System with a retention time of 365 days.

7.5 Example 5. Generation of regional grid surface dataset

A regular grid surface dataset can be generated at run time for a single gridcell or for gridcells comprising a regular regional or global domain. To generate a surface dataset for a regional run, the cpp tokens LSMLON and LSMLAT must be set to the desired resolution (e.g., LSMLON=1, LSMLAT=1 for a single point simulation) and the variables MKSRF_OFFLINE_EDGES, MKSRF_OFFLINE_EDGEE, and MKSRF_OFFLINE_EGDEW and their values need to be added to the namelist. A surface dataset will be created with the name surface-data.LSMLONxLSMLAT.nc (e.g., for a single point simulation the file name will be surface-data.001x001.nc). The model can then be run for the single point or for the regional domain by following Example 1 where FSURDAT is set to the new surface dataset.

In the following example, a regional grid is created over the Amazon. LSMLON and LSMLAT should be set to 15 and 11, respectively to obtain an regional surface dataset on a 3x3 grid.

&clmexp
CASEID = 'create_regional_surfdat'
NSREST = 0
NESTEP = 2
START_YMD = 19971231
DTIME = 1800
FSURDAT = ' ',
FRIVINP_RTM = '$CSMDATA/rtmdata/rdirc.05'
FPFTCON = '$CSMDATA/pftdata/pft-physiology'
OFFLINE_ATMDIR = '$CSMDATA/NCEPDATA'
MKSRF_OFFLINE_FNNAVORYO = '$CSMDATA/rawdata/mksrf_navyoro_20min.nc'
MKSRF_FVEGTYP = '$CSMDATA/rawdata/mksrf_pft.nc'
MKSRF_FSUITEX = '$CSMDATA/rawdata/mksrf_soitex.10level.nc'
MKSRF_FSOICOL = '$CSMDATA/rawdata/mksrf_soicol_clm2.nc'
MKSRF_FLANWAT = '$CSMDATA/rawdata/mksrf_lanwat.nc'
MKSRF_FGLACIER = '$CSMDATA/rawdata/mksrf_glacier.nc'
MKSRF_FURBAN = '$CSMDATA/rawdata/mksrf_urban.nc'
MKSRF_FLAI = '$CSMDATA/rawdata/mksrf_lai.nc'
FSURDAT = ' ' 
A surface dataset named surface-data.LSMLONXLSMLAT.nc will be created in the model executable directory. LSMLON and LSMLAT are defined in jobscript.csh (see 3.1.3).

MKSRF_OFFLINE_FNAVYRO = '$CSMDATA/rawdata/mksrf_navyoro_20min.nc'
Points to the orography dataset used to derive the model's land mask in offline mode. The environment variable $CSMDATA is explained in 3.1.1.

MKSRF_FVEGTYP, MKSRF_FSOITEX, MKSRF_FSOICOL, MKSRF_FLANWAT, MKSRF_FGLACIER, MKSRF_FURBAN, and MKSRF_FLAI
Specify the raw (usually high resolution) input datasets used to create the model surface dataset.

MKSRF_OFFLINE_EDGES, MKSRF_OFFLINE_EDGEN, MKSRF_OFFLINE_EDGEE, and MKSRF_OFFLINE_EDGEW
Must be defined for the desired model regional domain. The units are degrees north for edges and edgen and degrees east for edgee and edgew.

7.6 Example 6. Generation of global gaussian surface dataset

Only global surface datasets can be created on a non-regular grid, such as a gaussian grid. To generate a surface dataset on a gaussian grid, the cpp tokens LSMLON and LSMLAT must be set to the desired resolution (e.g., LSMLON=128, LSMLAT=64 for a T42 grid, and MKSRF_OFFLINE_FGRID must be set to the appropriate dataset in $CSMDATA/rawdata specifying the model grid, land mask and land fraction for the model grid. At T42 resolution, a surface dataset, "surface-data.128x064.nc", will be created in the model executable directory. This dataset may be renamed by the user to be more self-explanatory.

The following namelist will result in the generation of a surface dataset on a global gaussian grid.

&clmexp
CASEID = 'create_global_surfdat'
NSREST = 0
NESTEP = 2
START_YMD = 19971231
D TIME = 1800
FSURDAT = ' ' 
FRIVINP_RTM = '$CSMDATA/rtmdata/rdirc.05'
FPFTCON = '$CSMDATA/pftdata/pft-physiology'
OFFLINE_ATMDIR = '$CSMDATA/NCEPDATA'
MKSRF_OFFLINE_FGRID = '$CSMDATA/rawdata/T42_clm2_camfgrid_040802.nc'
MKSRF_FVEGTYP = '$CSMDATA/rawdata/mksrf_pft.nc'
MKSRF_FSOITEX = '$CSMDATA/rawdata/mksrf_soitex.10level.nc'
MKSRF_FSOICOL = '$CSMDATA/rawdata/mksrf_soicol_clm2.nc'
MKSRF_FLANWAT = '$CSMDATA/rawdata/mksrf_lanwat.nc'
MKSRF_FGLACIER = '$CSMDATA/rawdata/mksrf_glacier.nc'
MKSRF_FURBAN = '$CSMDATA/rawdata/mksrf_urban.nc'
MKSRF_FLAI = '$CSMDATA/rawdata/mksrf_lai.nc'
/
A surface dataset named surface-data.128x064.nc will be created at run time in the model executable directory.

**FSURD**

A surface dataset named surface-data.128x064.nc will be created at run time in the model executable directory.

**MKSRF_OFFLINE_FGRID** = '$CSMDATA/rawdata/T42_clm2_camfgrid_040802.nc'

Points to the dataset containing the model grid, land mask and fractional land for the surface dataset.

**MKSRF_FVEGTYP**, **MKSRF_FSOITEX**, **MKSRF_FSOICOL**, **MKSRF_FLANWAT**, **MKSRF_FGLACIER**, **MKSRF_FURBAN**, and **MKSRF_FLAI**

Specifies the input datasets used to create the surface dataset.

### 8 Testing Model Changes

The script `test-model.pl` in the `bld/offline/tests` directory runs a suite of basic tests for the CLM2.1 model running in offline mode on a T42 model grid. In order to use `test-model.pl`, users need to have at least Perl version 5.004 on their system. This test suite is designed for model testing during the modification of model code or for simply porting the code to another machine. The test suite provides a check that the basic functionality of the model still works despite the changes that have been introduced.

In this section, usage of `test-model.pl` for basic acceptance testing is described. The command-line arguments to “test-model.pl” and environment variables useful to the script are described. The script is designed to be run interactively. Although this script can be run at other labs, testing has only been done on NCAR machines. Consequently, the following discussion will be confined to NCAR platforms. The user should feel free to try the scripts at other labs.

Before running `test-model.pl`, the user needs to set the environment variable, **CSMDATA**, to the full disk pathname containing the un-tarred data subdirectories. At NCAR, CCSM input datasets are stored on a NFS mounted directory, “/fs/cgd/csm/inputdata/lnd/clm2”. Consequently, this is the default value for **CSMDATA** in the testing scripts. If the user has un-tarred the source code such that the `test-model.pl` script is in the directory “/home/user/clm2/bld/offline/tests”, then the user may invoke the script as follows:

```bash
cd /home/user/clm2/bld/offline/tests
test-model.pl
```

The script, `test-model.pl`, is designed such that the common settings the user might want to control are easily set by command line arguments. The “-h” option to `test-model.pl` lists all of the possible command-line options.

- `-h` = Help (this message)
- `-t` = List the tests that are performed
- `-clean` = Clean the old directories out
- `-nofail` = Continue even if errors are found
- `-res` = Resolution (T42)
- `-s n` = Skip to given test n (or range of numbers)
- `-c dir` = Compare to another version of the CLM2 model in this directory
- `-l lab` = Set the lab you are running at

The “-t” option to “test-model.pl” lists all of the possible tests. Below is a brief summary of each test.
• Tests 1 and 2. Run three time-steps with DEBUG compiler flags on, with SPMD on and off, and then ensure that answers are identical.

• Tests 3, 4, and 5. Perform an initial run followed by a restart using fewer SPMD tasks (if SPMD enabled), and fewer threads. Then do an initial run the same number of timesteps as the restart, and compare answers ensuring they are identical.

• Test 6. If a comparison to a previous code library is requested (via the "-c" option), repeat the last initial run with the previous code library and check if answers are identical.

An important feature of test-model.pl is the ability to compare user modified model code to a previous program library. This is useful in order to ensure that the implemented modifications do not change answers if that is what is expected. Using the command line option "-c dir" the user can compare to a previous program library by giving the full path to the root of the library to compare to. For example, if a test library can be found in "/home/user/clm2mod/src" and the unmodified library is located in "/home/user/clm2/src" then "-c" can be used as follows:

    cd /home/user/testmod/bld/offline/tests
test-model.pl -c /home/user/clm2/src

Many times differences with respect to a control library are intended to be bit-for-bit. When the "-c" option is used, only test 6 is compared to test 5 (see the list of tests above). If this comparison is identical, the modified model is identified as being bit-for-bit with the control library. Whereas test-model.pl can easily identify if two model libraries give identical answers, it is more difficult to verify if changes are within machine roundoff. Currently, if non bit-for-bit differences occur, differences between the history files produced by both the control as well as modified libraries must be examined to determine if these differences indicate only roundoff level changes.