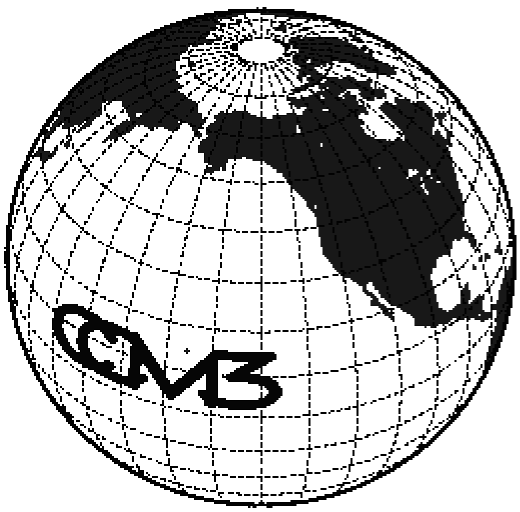
User’s Guide to NCAR CCM3.6

Erik B. Kluzek
Jerry Olson
James M. Rosinski
John E. Truesdale
Mariana Vertenstein



Climate and Global Dynamics Division

National Center for Atmospheric Research

Boulder, Colorado

NCAR TECHNICAL NOTES

The Technical Note series provides an outlet for a variety of NCAR manuscripts that contribute in specialized ways to the body of scientific knowledge but which are not suitable for journal, monograph, or book publication. Reports in this series are issued by the NCAR Scientific Divisions; copies may be obtained on request from the Publications Office of NCAR. Designation symbols for the series include:

EDD -- Engineering, Design, or Development Reports

Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

IA -- Instructional Aids

Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

PPR -- Program Progress Reports

Field program reports, interim and working reports, survey reports, and plans for experiments.

PROC -- Proceedings

Documentation of symposia, colloquia, conferences, workshops, and lectures. (Distribution may be limited to attendees.)

STR -- Scientific and Technical Reports

Data compilations, theoretical and numerical investigations, and experimental results

The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Acknowledgments

The authors would like to express appreciation to those members of the Climate Modeling Section in the Climate and Global Dynamics Division at NCAR who reviewed this User's Guide. We especially would like to thank Dave Williamson and Bill Collins.

Table of Contents

User's Guide to NCAR CCM3.6 1

Acknowledgments i

Table of Contents ii

List of Figures iv

List of Tables v

1. Introduction 1
- 1.1 What is new with CCM3.6 1
- 1.2 Overview 3

2. Using CCM3.6 5
- 2.1 How to Build and Run CCM3.6 5
  - 2.1.1 Obtaining the Source Code and Datasets 5
  - 2.1.2 Creating and Running the Executable 7
  - 2.1.3 Running the Distributed Memory Implementation 14
- 2.2 Model Input Variables 15
- 2.3 Model Input Datasets 25
  - 2.3.1 Atmosphere Component Datasets 26
  - 2.3.2 Ocean Component Datasets 28
  - 2.3.3 Land Surface Model Input Datasets 29
- 2.4 Example Namelists 31
  - 2.4.1 Example 1 -- Initial Run, One Day 32
  - 2.4.2 Example 2 -- Initial Run, T21 35
  - 2.4.3 Example 3 -- Restart Run, 30 Days 37
  - 2.4.4 Example 4 -- Paleo run 39
  - 2.4.5 Example 5 -- Branch Run 41
  - 2.4.6 Example 6 -- Initial Run, with additional fields 43
  - 2.4.7 Example 7 -- Run with Multiyear SST Dataset 44
  - 2.4.8 Example 8 -- Generate a Monthly-Averaged History File 46
  - 2.4.9 Example 9 -- Debugging Run 47
- 2.5 Model Output Datasets 49
  - 2.5.1 Model History Files 49
  - 2.5.2 Restart Datasets 59
  - 2.5.3. Mass Store Archiving. 60
- 2.6 Getting Help 60
  - 2.6.1 The CCM Web Page 60
  - 2.6.2 The ccm-users e-mail Group 61

3. CCM3.6 Internals 63
- 3.1 Design Philosophy 63
- 3.2 Model Code Flow 64
  - 3.2.1 Initialization 65

3.2.2 Time Integration 65
3.3 Data Structures 68
3.3.1 Vertical Coordinate 68
3.3.2 Gridpoint Data Structures 70
3.3.3 Spectral Data Structures 79
3.3.4 Other Common Blocks and Header Files: The Parameterization Interface 79
3.4 Multitasking Strategy 81
3.4.1 Shared-Memory Multitasking 82
3.4.2 Distributed Memory Multitasking 84
3.5 Output Datasets 85
3.5.1 History Files 86
3.5.2 Restart Files 99
3.6 Disk and Mass Store File Management 100
3.6.1 Disk File Management 101
3.6.2 Use of the NCAR Mass Storage System 101
3.6.3 Issuing Shell Commands from the Model 102

4. Changing the Model 103
4.1 Modifying the Code 103
4.2 Adding New Variables 103
4.3 Adding New Constituents 105
4.3.1 Initializing New Constituent Data Fields 106
4.3.2 Surface Fluxes, Sources, and Sinks 107
4.3.3 Adjustment Processes 107
4.3.4 Global Minima 107
4.4 Changing Resolution 108
4.5 Adding Parameterizations 108
4.6 Trouble-Shooting Model Changes 108
4.6.1 Resource Allocation Errors 109
4.6.2 Coding Errors 110
4.6.3 Formulation Errors 110

References 113

Appendix A: Glossary 115

Appendix B: CCM3.6 Calling Tree 119

Appendix C: Model Error Messages 125

List of Figures

Figure 1.1: Conceptual flowchart of the code. 3

Figure 3.1: Hybrid vertical coordinate. 68

Figure 3.2: Layout of main model buffers. 71

Figure 3.3: Use of SSD work units. 75

Figure 3.4: /com3d/ storage arrangement. 76

Figure 3.5: Spectral storage arrangement when running on a parallel vector processor. 79

Figure 3.6: Spectral storage arrangement when not running on a parallel vector processor. 80

Figure 3.7: History file format. 86

Figure 3.8: A typical history file latitude record. 93

Figure 3.9: History buffer layout. 95

Figure B: Flowchart 119

List of Tables

Table 2.1: Directory hierarchy	5
Table 2.2: `misc.h` pre-processor tokens	10
Table 2.3: `params.h` pre-processor tokens	11
Table 2.4: `preproc.h` pre-processor tokens	12
Table 2.5: `CCMEXP` namelist Input Variables	16
Table 2.6: `LSMEXP` namelist input variables	24
Table 2.7: Atmospheric component initial dataset fields	26
Table 2.8: Additional Slab Ocean Model initial dataset fields	27
Table 2.9: Slab Ocean Model time-varying boundary dataset fields	29
Table 2.10: Land Surface Model: initial dataset time invariant fields	29
Table 2.11: Land Surface Model: initial dataset time variant fields	30
Table 2.12: Land Surface Model: resolution independent datasets	31
Table 2.13: Master field list	50
Table 2.14: Additional fields on a monthly-averaged primary history file	57
Table 3.1: History file header record 1-- integer variables	88
Table 3.2: History file header record 2 -- character variables	90
Table 3.3: History file header record 3 -- real variables	92
Table 3.4: Description of field information list arrays	92
Table 4.1: SI Units of History File Fields	104

1. Introduction

This guide was written with two goals in mind: (1) to instruct the novice user on running CCM3.6 and (2) to provide a description of the code organization and data structures. In the appendices we have included a glossary of terms and abbreviations, model code flow, and output error messages.

1.1 What is new with CCM3.6

CCM3.6 is the latest version of the NCAR Community Climate Model. It is in the same family as CCM3.0 (Acker et. al., 1996) and therefore most of the code is the same with only minor changes to the physical parameterizations. None of the modifications significantly alter the simulation climate from that produced by CCM3.2, provided, of course, that the input namelist variables are appropriately chosen for a CCM3.2-like run. Modifications to CCM3.2 include:

- **Physics changes (no significant change to the climate):**
    - Added capability to calculate earth's orbital parameters. Changed algorithm to calculate solar zenith angle (Berger, 1977).
    - Modified surface stress calculation.
    - Solar radiation calculations now include 19 bands. This allows comparison to Nimbus-7 spectral albedos (Collins, 1998).
    - 2m-reference height temperature computed everywhere, both in flux-coupled mode and stand-alone mode.
    - Snow capped to 1m over land ice.
    - LSM now conserves mass globally.
- **Changes to input/output files:**
    - The ozone dataset is now interpolated horizontally and vertically during the model run.
    - All input files are now netCDF format rather than ccm history file format.
    - LSM and CCM generate initial conditions files during model runs.

- Input and output LSM datasets are in netCDF format.
- LSM can now generate its own initial dataset at model startup.

- **Changes to user interface:**
  - Added capabilities to run model either adiabatically or with prescribed heating and temperature profiles.
  - Solar constant added as a namelist input variable.
  - Volume mixing ratios for greenhouse gases added as namelist input.
  - Output primary history files and restart datasets renamed to correspond to model date.
  - Added a "restart-pointer file" to keep track of the latest restart file written. This way the user doesn't have to figure out the name of the last restart file written to restart a run.

- **Bug fixes (no significant change to the climate):**
  - Calendar no longer off by 1 day.
  - Fixed vector temporary that was improperly used outside of loop in radiation code (routine `radabs`).
  - Corrected pressure levels used in routine `difcor`.
  - Corrected minor problem with top of atmosphere radiation calculation (routine `radtpl`).
  - Corrected minor problem with water vapor paths used in trace gas overlap (routine `trcabn`).
  - Fixed potential out of bounds memory reference in buoyancy calculation (routine `bouyan`).
  - Workaround added to LSM to address Cray compiler bug. See the CCM3 web-site for more information on this at:
    http://www.cgd.ucar.edu:80/cms/ccm3/news/ccm3.2_fix.html.

- **Recently Identified Bug:**

- Overestimate of large-scale generation of CAPE in the sub-cloud layer. A patch and description is available on the CCM3 web page. This patch does significantly alter the simulated climate.

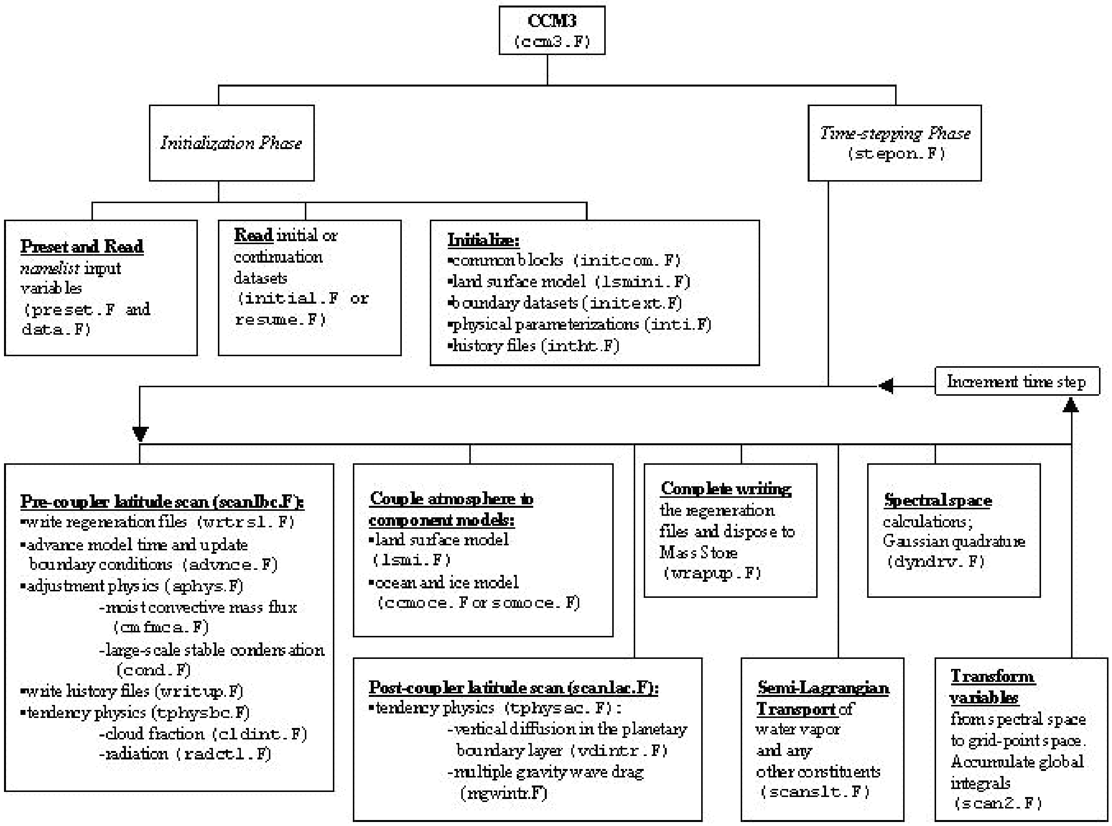
Code and documentation for the Slab Ocean Model (SOM) and the Semi-Lagrangian Dynamics (SLD) are provided as an unsupported feature of the distribution. These features will be supported in future releases.

1.2 Overview

The basic structure of CCM3.6 includes an initialization phase and a time integration phase. A conceptual flowchart with functional description of each phase and a list of relevant routines is shown below. This figure also serves as a reference linking key elements of the CCM3.6 physics (Kiehl et. al., 1996) to the code.   A calling tree, showing all program modules except single-purpose utility routines, is provided in "Appendix B: CCM3.6 Calling Tree".

Most of the source code files contain one subroutine or function per file where the file name reflects the subroutine name. Exceptions to this are in the "csm_share" sub-directory where each file contains multiple subroutines (however the subroutine names all start with the filename as a pre-fix). A ".F" (Fortran code) or a ".c" (C code) suffix denotes source code files. The capital "`F`" suffix indicates that the Fortran source contains preprocessing directives. Files named with a "`.h`" suffix are header files that contain preprocessing directives, variable declarations, parameter definitions, and common block definitions. Contents of select header files are included via *#include* statements at the beginning portion of each of the "`.F`" and "`.c`" files.

Figure 1.1 Conceptual flowchart of the code.



The initialization phase is responsible for three activities. First, the namelist input variables, are preset and read. Next, the initial dataset is read (or continuation datasets, depending on the type of run). This is followed by a series of initializations required to prepare the data structures, output files, etc., for the time integration phase. The time integration phase solves the equations describing the atmospheric general circulation. This phase includes gridpoint physics and dynamics, semi-Lagrangian transport, spectral space calculations, and a conversion between spectral space and grid-point space.

2. Using CCM3.6

2.1 How to Build and Run CCM3.6

2.1.1 Obtaining the Source Code and Datasets

It is assumed that the user has access to the utilities UNIX **tar** and Free Software Foundation **gunzip** and **gnumake**. The source code along with input datasets are located on the NCAR anonymous **ftp** server ftp.ucar.edu (IP address 128.117.64.4) in subdirectory `ccm/`. These files are also available on the World Wide Web under the URL:

`http://www.cgd.ucar.edu/cms/ccm3/index.html`

A sample ftp session might look like:

---

ftp ftp.ucar.edu
Name: anonymous
password: username@domain
cd ccm
bin
get ccm3.6.code.tar.gz
get ccm3.6.datasets.tar
quit

Note that **username@domain** represents the email address of the person obtaining the CCM3.6 files, and **bin** instructs **ftp** to set a binary transfer.

The file `ccm3.6.code.tar.gz` contains code, documentation and build scripts. This file must first be uncompressed with the GNU **gunzip** utility and then "untarred" as follows:

gunzip -c ccm3.6.code.tar.gz | tar xvf -

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

Table 2.1: Directory Hierarchy

Directory name	Synopsis
bld/	Scripts to build and execute the model
src/	Directory of FORTRAN and "C" source code
src/physics	Physics routines (e.g., radiation, convection)
src/control	Control code
src/ccmlsm_share	Code shared between the LSM and the CCM (e.g., getting mass store files, calculating dates)
src/dynamics	Dynamics routines shared by both Eulerian and Semi-Lagrangian dynamics
src/dynamics/eul	Eulerian dynamics
src/dynamics/sld	Semi-Lagrangian dynamics
src/spmd	Distributed-memory message passing implementation shared by both Eulerian and Semi-Lagrangian dynamics
src/spmd/eul	Distributed-memory message passing implementation of Eulerian dynamics
src/spmd/sld	Distributed-memory message passing implementation of Semi-Lagrangian dynamics
src/lsm	Land Surface Model
src/som	Slab Ocean Model
src/dom	Data Ocean Model
src/csm_share	Code shared by all the geophysical model components of the Climate System Model (CSM)
src/csm_share/msg	Code for CSM message passing
src/mathutil	Math utilities needed for non-NCAR machines (primarily

	the FFT library)
src/srchutil	Search utilities needed for non-CRAY machines (corresponding to CRAY-intrinsic array search utilities)
doc/	Documentation for CCM3.6 and the Land Surface Model
data/	Directory where datasets should be untarred
tools/	Directory of tools for input dataset manipulation
tools/interpic	Code for interpolating initial condition datasets both vertically and horizontally
tools/definesurf	Code to create the required input fields `ORO`, `PHIS` and `SGH` on an existing initial condition dataset
tools/interpsst	Code for creating resolution specific input SST datasets

2.1.2 Creating and Running the Executable

CCM3.6 may be built to run in one of two modes. It can run as a stand-alone executable where the Land Surface Model (LSM) and the ocean component are contained as part of the executable (see the "LSM User's Guide" for a complete description of using the LSM). This will be referred to as stand-alone mode. In this mode, the model's ocean component can be run in one of two ways: in data ocean or slab ocean mode. The data ocean configuration uses climatological sea surface temperatures, whereas the slab ocean configuration uses data for ocean mixed layer depth and ocean mixed layer heat flux.

Alternatively, CCM3.6 can be run as a component in a system of geophysical models (CSM) in which two-way interaction is enabled through the use of a flux coupler (Bryan *et al.*, 1996). In this mode, the LSM, ocean and sea-ice models are run as separate executables that communicate with the atmospheric component via the flux coupler. This will be referred to as flux-coupled mode. The generation and execution of a flux-coupled executable will not be addressed in this document. A separate document addressing these issues is forthcoming.

To create a stand-alone executable, the user must build the model for the desired target architecture, type of model dynamics, ocean configuration, and if applicable, distributed memory (SPMD) implementation. Three target architectures are supported: Cray, SGI and Sun. Two types of dynamics are possible, Eulerian or Semi-Lagrangian (the latter, however, is still experimental and will be supported in later releases). Furthermore, either data ocean or slab ocean configurations can be used. Finally, the distributed memory implementation can only be used on non-Cray architectures (see "Running the Distributed Memory Implementation").

All files relating to the creation and execution of the stand-alone executable are contained in the `ccm/bld` directory. There are four architecture-dependent scripts in this directory which provide step-by-step instructions for building and running the executable. These build scripts create a model executable, determine the input datasets and construct the input model namelists. It is assumed that the user will run the appropriate build script from the `ccm/bld` directory and that all input datasets have been untarred . The build script will then create the directory `ccm/run`. The build scripts can be run with minimal user modification, **assuming the user has untarred the input datasets in the directory** `ccm/data.` These scripts are provided as examples to help the novice user get CCM3.6 up and running as quickly as possible. Users who are already familiar running the CCM will want to modify these scripts or use their build procedure. The default model configuration resulting from each build script is summarized below.

Script name	Synopsis
build.cray_ncar.csh	T42 resolution, Eulerian dynamics, data ocean component, no SSD, NCAR Cray computers
build.cray_nonncar.csh	T42 resolution, Eulerian dynamics, data ocean component, no SSD, non-NCAR Cray computers
build.noncray.csh	T42 resolution, Eulerian dynamics, data ocean component, SGI or Sun computers
Build.noncray_spmd.csh	T42 resolution, Eulerian dynamics, data ocean component, distributed memory (SPMD) implemented, SGI or Sun computers

Each script can be divided into four functional sections: 1) specification of environment variables; 2) creation of three header files and a directory search path file needed to build the model executable; 3) creation of the model namelist; and 4) execution of the model.

2.1.2.1 Specification of environment variables

The following environment variables must be set from within the build script.

Environment Variable	Synopsis
`MODEL_SRCDIR`	The full pathname for the source code directory hierarchy. Default setting is full pathname for `ccm/src`
`MODEL_EXEDIR`	Full pathname for the directory where the model executable will reside (object files will be built in the directory `$MODEL_EXEDIR/obj`). Default setting is full pathname for `ccm/run`

MODEL_DATDIR	Full pathname for the directory where the input datasets may reside (see "Model Input Datasets"). Default setting is full pathname for ccm/data
MODEL_BLDDIR	Full pathname for the directory where the **gnumake** makefile resides. Default setting is full pathname for ccm/bld.
CPU	Target architecture. Must be set to CRAY,SGI or SUN (in uppercase). Default setting for Cray build scripts is CRAY. Default setting for non-Cray build scripts is SGI.
NCAR_CRAY	Additional environment variable needed if running on a Cray. Should be set to TRUE if running on an NCAR CRAY, otherwise should be set to FALSE. Set automatically by Cray build scripts.
SSD	Additional environment variable needed if running on a Cray. Should be set to TRUE if using the solid-state storage device (SSD), otherwise should be set to FALSE. Default setting is FALSE
LIB_NETCDF	Full pathname for directory containing the netCDF library. Default setting is /usr/local/lib
INC_NETCDF	Full pathname for directory containing netCDF include files. Default setting is /usr/local/include
MAXCPUS	Maximum number of processors used for multitasking. Should not exceed the number of physical CPU's on the machine. Default setting is 16 for Crays and 2 for non-Crays.
LIB_MPI	Full pathname for directory containing the MPI library. Default setting is /usr/local/lib
INC_MPI	Full pathname for the directory containing the MPI include files. Default setting is /usr/local/include

2.1.2.2 Creation of header files and directory search path

Using the values of the above environment variables, the build script creates the header files **misc.h**, **params.h**, and **preproc.h** and the directory search path file **Filepath**. These files are placed in the directory $MODEL_EXEDIR/obj. To modify these files the user should edit their contents from within the build script rather than attempt to edit the files directly since the build script will overwrite the files upon its execution. The use of these files by **gnumake** is discussed in "Details of gnumake procedure". The contents of each of these files are summarized below.

- **Filepath**

  Contains a list of directories used by **gnumake** to resolve the location of source files and to determine dependencies. Contents will vary with target architecture, choice of model dynamics, ocean configuration and use of distributed memory. Examples of **Filepath** contents for different model configurations can be found in **ccm/bld/example_Filepath**.

- **misc.h**

  Contains a list of resolution-independent `cpp` directives. The following `cpp` tokens (except for `NCPREC`) must either be defined or undefined (i.e. preceded by `#define` or `#undef`). An example of a **misc.h** file produced by the build script can be found in **ccm/bld/example_misc.h**.

Table 2.2: misc.h pre-processor tokens

misc.h cpp tokens	**Synopsis**
`CRAY,SUN,SGI`	Platform type. Only one of these can be defined. Default setting is `CRAY` for Cray scripts and `SGI` for non-CRAY scripts.
`COUP_CSM`	Define if the flux-coupled ocean configuration will be used. `COUP_CSM` **is always undefined for stand-alone mode**.
`COUP_SOM`	Define if the slab ocean model is to be used. The data ocean component is invoked by making this token undefined. Default setting is with `COUP_SOM` undefined (run with DOM).
`SHELL_MSS`	Define where subroutine interface for `mswrite` does not exist. Default setting is that `SHELL_MSS` is not defined for NCAR Cray and defined otherwise.
`FORTFFT`	Must be defined when using the Fortran version of the ECMWF FFT package. This `cpp` token provides the additional workspace needed by the Fortran FFT code. Default setting is to not define `FORTFFT` for NCAR Cray and define otherwise.

PVP	Parallel vector processor. Default setting is to define PVP for CRAY architectures but leave it undefined otherwise.
NCPREC	Precision of output netCDF history files. Set to NF_FLOAT (single-precision output) or NF_DOUBLE (double-precision output) for output LSM netCDF history files. Default setting is NF_FLOAT.
REALTYPE	Must be set only for distributed-memory implementation. Default setting is MPI_DOUBLE_PRECISION.
SPMD	Enables distributed memory (SPMD) implementation on non-Cray architectures. Refer to the "Running the Distributed Memory Implementation" .

- **params.h**

  Contains a list of resolution-dependent cpp directives for non-LSM routines. Examples of various **params.h** contents produced by the build script for different model resolutions can be found in **ccm/bld/example_params.h**.

Table 2.3: params.h pre-processor tokens

params.h cpp tokens	Synopsis
PCNST	Number of advected constituents including water vapor. Default setting is 1 (for water vapor).
PNATS	Number of non-advected constituents. Default setting is 0.
PLEV	Number of vertical levels. Default setting is 18.
PLEVR	Number of vertical levels over which radiation calculations are performed. Use of separate vertical coordinate for the radiation calculation has not been tested. Therefore this directive must currently be set to the same value as PLEV.
PLON	Number of longitudes on the transform grid.

	Default setting is 128.
PLAT	Number of Gaussian latitudes on the CCM3.6 transform grid. Default setting is 64.
PTRM	Spectral truncation of the zonal wavenumber `m`. Default setting is 42.
PTRN	Spectral truncation of the total wavenumber `n` for zonal wavenumber 0. Default setting is 42.
PTRK	Maximum total wavenumber `k`, for any zonal wavenumber `m`. Default setting is 42.
POZLEV	Number of ozone levels on the input ozone dataset. Default setting is 23.

- **preproc.h**

  Contains a list of resolution-dependent **cpp** tokens for the LSM code. Currently the resolution of the LSM **must** be the same as that of the atmospheric component. A typical **preproc.h** specification can be found in **ccm/bld/example_preproc**.

Table 2.4: preproc.h pre-processor tokens

preproc.h cpp tokens	Synopsis
LSMLON	Number of longitudes for the LSM grid. Currently this directive must have the same value as `PLON` in **params.h**.
LSMLAT	Number of latitudes for the LSM grid. Currently this directive must have the same value as `PLAT` in **params.h**.

2.1.2.3 Details of gnumake procedure

`cpp` directives of the form `#include`, `#if defined`, etc., are used to enhance portability, and allow for the implementation of distinct blocks of platform-specific code within a single file. Header files, such as **misc.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. `cpp` directives are also used to perform textual substitution for resolution-specific parameters in the code. The format of these `cpp` 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(plat = PLAT)
```

will result in the following processed line (for standard T42 resolution).

```
parameter(plat = 64)
```

gnumake generates a list of source and object files using each directory listed in **Filepath.** For each source file, **gnumake** invokes `cpp` to create an dependency file in the directory `$MODEL_EXEDIR/obj`. For example, `file.F` will have a dependency file, `file.d`. If a file listed as a target of a dependency does not exist in `$MODEL_EXEDIR/obj`, **gnumake** searches the directories contained in **Filepath**, in the order given, for a file with that name. The first file found satisfies the dependency. If user-modified code is to be introduced, **Filepath** should contain, as the first entry, the directory containing the user code (see "Modifying the Code").

A parallel **gnumake** is achieved in the build scripts by using **gnumake** with the **-j** option, which specifies the number of jobs (commands) to run simultaneously. Caution should be exercised on Cray machines, because if this number is set too high, it is possible to exceed memory and/or number of process limits. The current default value in the build script is 4.

2.1.2.4 Running the Model

In addition to building the model executable, the build scripts allow the user to define the appropriate input datasets, generate the model namelist and run the executable. Input datasets and model namelists are discussed in "Model Input Datasets" and "Model Input Variables", respectively. If the code is targeted to run multitasked on a CRAY PVP machine, the environment variable `$NCPUS` will also be set. Its value should not exceed the number of physical CPU's on the machine. The build script sets `$NCPUS` to the script environment variable, `$MAXCPUS`.

For CRAY architectures, the build script can either be run in batch or interactive mode. Since climate integrations are often run for long periods of time it may be more convenient in such cases to use a batch facility to complete the run. The following "**QSUB**" directives at the top of the **build.cray_ncar.csh** or **build.cray_nonncar.csh** script must be modified accordingly. Note that the following memory and SSD limits are sufficient for running at T42 horizontal resolution multitasked on eight processors. Refer to "Shared-Memory Management" for more information on memory requirements.

NQS setting	Synopsis
#QSUB -q reg	Job class, regular
#QSUB -lT 600	CPU time limit for one simulated day, 600 seconds
#QSUB -lM 37Mw	Main memory requirement without SSD, 37 megawords (comment if SSD is to be used)

#QSUB -lM 23Mw	Main memory requirement with SSD, 23 megawords (uncomment if SSD is to be used)
#QSUB -lQ 14Mw	SSD memory requirement, 14 megawords (uncomment if SSD is to be used)
#QSUB -eo	Directs stdout and stderr to the same file
#QSUB -s /bin/csh	Selects shell
#QSUB -x	Export environment variables

The script may then be submitted to the batch facility NQS (Network Queuing System) via the command

qsub build.cray_ncar.csh

2.1.3 Running the Distributed Memory Implementation

This section describes how to run CCM3 under MPI on the SUN and SGI architectures (message passing is not required to run on these architectures). It is written for the user who is already knowledgeable about MPI operation and terminology. It is also assumed that MPI is built on the host machines. Information on how to configure, build, and run MPI is freely available via the web.

Running CCM3 under MPI on the SUN and SGI architectures consists of two steps: building the MPI executable and starting the model using the appropriate command line arguments for MPI. Configuring CCM3.6 for MPI requires defining the SPMD token in the header file `misc.h` and including the directories `$MODEL_SRCDIR/spmd` and either `$MODEL_SRCDIR/spmd/eul` or `$MODEL_SRCDIR/spmd/sld` in **Filepath**. Additionally, the directories containing the MPI library and MPI include files must be specified as environment variables in the build script.

Most MPI implementations provide a startup script called `mpirun`, 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. For example the command used to run the MPI version using two processes might be:

mpirun -np 2 atm < atm.parm

assuming that `atm.parm` corresponds to the input namelist file and `atm` to the executable.

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. A few caveats should be kept in mind when running the MPI version of the model.

- The user must specify the number of processes to use. This is done via the `-np` parameter to `mpirun`.
- The user must specify an even number of processes.
- Since MPI creates remote processes, the user must take care that process limits for non-interactive shells are sufficient for model size. The Sun default stack size of 8Mb is one such limit, which can cause higher resolution model runs to fail, usually with a segmentation fault.
- If the executable is not on a shared file system, you must copy the executable to each host manually.

2.2 Model Input Variables

A model run is controlled using the namelists `CCMEXP` and `LSMEXP`. `CCMEXP` is the namelist for the atmospheric component and `LSMEXP` is the LSM namelist. The focus of this section is to describe the variables that constitute the namelist input. For specific namelist examples, see "Namelist Examples".

The `CCMEXP` namelist control variables are broken into the following categories:

- Define the type and length of run
- Input datasets
- File archive control
- History file options
- Restart file options
- Model time
- Physics controls
- Non-water constituents
- Memory controls
- Orbital parameters

Table 2.5 displays each of the variables contained in `CCMEXP`. Variables are grouped according to the categories listed above. Variable names displayed in **BOLD** font must be defined to run the model. Included in the table are the following pieces of information:

- Variable name.
- Variable type (*Character, Integer, Real, or Logical*). If the variable is of type character, then its declared length is displayed along with the variable type (e.g. `CTITLE` is 80 characters in length, therefore C*80 is entered in the "Type" column).
- Default variable value.
- Examples from Sec. 2.4 demonstrating usage.
- Variable description

Most of the variables in `CCMEXP` are single-valued, but some are array-valued.

Table 2.5: CCMEXP namelist Input Variables

Variable Name	Type	Default Value	See	Description
Define the type and length of run				
CASEID	C*8	Blanks	Ex.1	Case identifier. Saved in history file header record (refer to Table 3.1), and used as part of the MSS path name (see Sec. 2.5.3).
CTITLE	C*80	blanks	Ex.1	Case title.
NELAPSE	I	-999999999	Ex.3	Elapsed time in iterations (positive) or days (negative) to run. May be input instead of `NESTEP`.* REQUIRED if `NESTEP` not set.
NESTEP	I	-999999999	Ex.1	Ending timestep (positive) or ending day (negative) of run.* REQUIRED if `NELAPSE` not set.
NSREST	I	0	Ex.1 Ex.3 Ex.5	Run type: 0=initial 1=restart 3=branch

				(option 2 no longer exists)
**No usable default value exists for this variable. One of `NESTEP` or `NELAPSE` is required to be set. If both are set, then the value of `NESTEP` will override the value of `NELAPSE`.*				
Input datasets				
BNDTVO	C*80	blanks	Ex.1 Ex.2	Full pathname or filename of time-variant, ozone mixing ratio's boundary dataset (netCDF format). REQUIRED
BNDTVS	C*80	blanks	Ex.1 Ex.2 Ex.7	Full pathname or filename of time-variant, sea-surface temperature's boundary dataset (netCDF format). REQUIRED
DATADIR	C*80	'.'		Directory where input datasets may be found
NCDATA	C*80	blanks	Ex.1 Ex.2 Ex.7	Full pathname or filename of initial dataset (netCDF format). Required for an initial run (NSREST=0). REQUIRED
OZNCYC	L	.TRUE.		Flag for yearly cycling ozone dataset. If .FALSE., assume multiyear dataset.
SSTCYC	L	.TRUE.	Ex.7	Flag for yearly cycling SST dataset. If .FALSE., assume multiyear dataset.
Mass Store controls				
IRT	I	99999	Ex.1 Ex.2 Ex.6 Ex.9	Retention period in days for history files stored on Mass Store. If IRT=0, no history or restart files will be archived.
NSWRPS	C*8	blanks	Ex.1 Ex.5	Mass Store, write password, for all output datasets. Must remain the same throughout a case, or a continuation run will not update original history files.
RIRT	I	99999	Ex.6	Retention period for Mass Store restart files. Same defaults and

				maximums as IRT. If RIRT=0, then no restart files will be archived.
History file options				
AUXF (pflds* (ptapes-1))	C*8	blanks	Ex.5 Ex.8	Input array used to define the fields to be output on up to 5 auxiliary history files; see Sec. 2.5.
CPUCHEK	L	.FALSE.		If set, check after each history file write, to determine if there is sufficient CPU time remaining, to do another history file write. If there is not, stop.
EXCLUDE (pflds)	C*8	blanks	Ex.5	List of fields to exclude from the primary history file (these fields would otherwise appear by default on the primary history file; see Sec. 2.5).
INITHIST	C*16	'MONTHLY'		Generate initial dataset as auxiliary history file. Can be set to 'MONTHLY' , 'YEARLY', or 'NONE'. If set to 'NONE' then initial datasets will not be created. Must convert resulting file to netCDF format before using as initial conditions.
MFILT (ptapes)	I	5,5,5,...	Ex.1 Ex.5 Ex.9	Array of number of time samples to write to the primary and auxiliary history files (a time sample is the history output from a given timestep; several time samples can fit on one file).
NDENS (ptapes)	I	1,1,1,...	Ex.1 Ex.5 Ex.9	Array specifying packing densities (1,2 or 4) for primary and auxiliary history files. NDENS is only valid on an NCAR Cray. (see CRAYLIB documentation)
NHTFRQ (ptapes)	I	-24,0,0,..	Ex.1 Ex.5 Ex.9	Array of time sample, write frequencies, for primary and auxiliary history files. If NHTFRQ(i)=0, no file will be

				written. If NHTFRQ(i)0, frequency is input as number of timesteps. If NHTFRQ(i)<0, frequency is input as number of hours.
NINAVG (ptapes)	C*1	'A','A',...	Ex.5 Ex.6 Ex.8	Array of field type flags for primary and auxiliary history files: 'A'= averaged over nhtfrq(i) 'I'= instantaneous 'M'= point minimum over nhtfrq(i) 'X'= point maximum over nhtfrq(i) 'Q'= monthly-averaged; applies only to the primary history file. All fields on a history file are of the type indicated by ninavg(i) except variables ORO and PHIS. ORO and PHIS are always instantaneous.
NLFILT	L	.FALSE.		Flag to put one extra time sample on the first history file of current case. This aids in bookkeeping of long runs.
PRIMARY (pflds)	C*8	blanks	Ex.6	List of fields to include in primary history file (must be in Master Field List; these fields do not appear by default on the primary history file; see Sec. 2.5).
STFNUM	I	1		Start history file and restart file naming with this number (min=1; max=9999).
Restart file options				
MXSZRG	I	2**30		Maximum size allowed for restart dataset . If MXSZRG< 10^6 units are megabytes, otherwise in bytes.
NREFRQ	I	1		Frequency of restart dataset writes. **This variable can only be 1 or 0**. If 1, restart files are written and archived

				for every archive of the primary history file. If this variable is 0, then no restarts are written.
NREVSN	C*72	blanks	Ex.5	Full pathname of master restart file from which to branch.
REST_PFILE	C*80	blanks		Full pathname of restart pointer file.
Model time				
DTIME	R	1200.	Ex.1	Length of model timestep in seconds. CAUTION: Changing this variable directly impacts the physical parameterizations in the model and may impact the climate. Changing resolution usually requires a change in DTIME.
NNBDAT	I	-9999	Ex.1 Ex.7	Base date for run as yyyymmdd. If not input, set to NCDATE from initial dataset header.
NNBSEC	I	-9999	Ex.1 Ex.7	Seconds of base date. If not input, set to NCSEC from initial dataset header.
NNDBAS	I	-9999	Ex.1	Base model day for run. If not input, set to NDCUR from initial dataset header.
NNSBAS	I	-9999	Ex.1	Seconds of base model day for run. If not input, set to NSCUR from initial dataset header.
Physics controls				
ADIABATIC	L	.FALSE.		If true, do not run model physics. Run only the dynamical core (in this mode, water vapor is advected as a passive tracer).
CO2VMR	R	3.550e-4	Ex.4	CO_2 volume mixing ratio.
CH4VMR	R	1.714e-6	Ex.4	CH_4 volume mixing ratio.

N2OVMR	R	0.311e-6	Ex.4	N_2O volume mixing ratio.
F11VMR	R	0.280e-9	Ex.4	CFC_{11} volume mixing ratio.
F12VMR	R	0.503e-9	Ex.4	CFC_{12} volume mixing ratio.
DIF2	R	2.5E5 *		del^2 horizontal diffusion coefficient.
DIF4	R	1.E16 *		del^4 horizontal diffusion coefficient.
DIVDAMPN	R	0.0		Number of days (from nstep 0) to run divergence damper. Use only if model is blowing up because of spin-up problems. Suggested value: 10.
EPS	R	0.06 *		Time filter coefficient.
FLXAVE	L	.TRUE.		If true, only send data to the flux coupler on radiation time steps. This namelist variable only relevant for flux-coupled runs.
IDEALDYN	L	.FALSE.		If true, only the dynamical core of the model is run. The model physics is replaced by an "idealized physics" routine, which specifies the heating and temperature profiles for the model climate. Water vapor is initialized to 0.
IRADAE	I	-12		Frequency of absorptivity/emissivity calculations in iterations (if positive) or model hours (if negative). To avoid having the abs/ems values saved on the restart output, this variable should divide evenly into MFILT(1)*NHTFRQ(1).
IRADLW	I	-1		Frequency of long-wave radiation calculation in timesteps (if positive) or model hours (if negative).
IRADSW	I	-1		Frequency of short-wave radiation calculation in timesteps (if positive) or model hours (if negative).

ITSST	I	1		Frequency of SST update in timesteps.
KMXHDC	I	1		Number of levels over which to apply Courant limiter, starting at top of model.
NLVDRY	I	3		Number of layers from the top of the model over which to do dry convective adjustment. Must be less than plev (the number of vertical levels).
PERTLIM	R	0.		Maximum size of perturbation to apply to initial temperature field (set non-zero for, numerics testing purposes only).
SCON	R	1.367E6	Ex.4	Solar constant.
TAUVIS	R	0.14		Visible optical depth.
* *Appropriate for the T42/18-level resolution*				
Non-water constituents				
NUSR_ADV	I	0		number of user defined advected tracers
NUSR_NAD	I	0		number of user defined non-advected tracers
READTRACE	L	.TRUE.		If true and PCNST+PNATS .GT. 1, obtain all initial tracer data from initial conditions dataset.
TRACE_TEST1 TRACE_TEST2 TRACE_TEST3	L L L	.FALSE. .FALSE. .FALSE.		If true, turn on test tracer code with 1 tracer (PCNST=2). If true, turn on test tracer code with 2 tracers(PCNST=3). If true, turn on test tracer code with 3 tracers(PCNST=4). NOTE: For testing/debugging purposes only. Leave FALSE during production runs.

Memory controls				
INCORBUF	L	.TRUE.	Ex.1	If true, then keep main model buffers in-core (buffers b1 and b2).
INCORHST	L	.TRUE.	Ex.1	If true, then keep the history buffer in-core.
INCORRAD	L	.TRUE.	Ex.1	If true, then keep abs/ems buffer in-core.
LINEBUF	L	.FALSE.	Ex.9	If true, force buffer flush of stdout with each new-line generated (useful for debugging).
Orbital parameters				
ECCEN	R	UNDEFINED	Ex.4	Earth's eccentricity of orbit. (unit-less: typically 0. to 0.1). REQUIRED
IYEAR_AD	I	UNDEFINED	Ex.1 Ex.4	Year (AD) used to compute earth's orbital parameters. If not set, then use the values from the eccen, mvelp, and obliq namelist parameters. REQUIRED
MVELP	R	UNDEFINED	Ex.4	Earth's moving vernal equinox at perihelion (degrees: 0. to 360.0). REQUIRED
OBLIQ	R	UNDEFINED	Ex.4	Earth's orbital angle of obliquity (degrees: -90. to +90., typically 22. to 26.). REQUIRED
If only IYEAR_AD is set, orbital parameters will be computed automatically (based on Berger, 1977). *If one of OBLIQ, ECCEN, OR MVELP is set, all three must be set.* *If all four of the above are set by the user, IYEAR_AD takes precedence.*				

The namelist LSMEXP is used to set variables that control the LSM portion of the model. Namelist variables are preset and read by routine lsmctli. Table 2.6 displays the LSMEXP variables. Note that several LSMEXP namelist variables are identical to and superceded by those in the CCMEXP namelist when CCM3.6 is run as a stand-alone model. These variables can only have values different from CCM3.6 when the LSM model is run through the flux coupler. Hence, they are not repeated in Table 2.6.

Table 2.6: LSMEXP namelist Input Variables

Variable Name	Type	Default Value	See	Description
ANTARTICA	L	.TRUE.		If true, extend Antarctica for Ross Ice Shelf glacier. Only used when FINIDAT is set to "arbitrary initialization"
CHNTYP (2,malflds)	C*8	Blanks		Paired 8-character field name and field type. Overrides, default settings.
CTITLE	C*80	Blanks	Ex.1	Case title.
DATADIR	C*80	'.'		Directory where input datasets reside.
EXCL (malflds)	C*8	Blanks		Names of fields to exclude from history file.
FINIDAT	C*80	'not_specified'	Ex.1 Ex.7	Initial conditions filename. REQUIRED
FLDS1D (malflds)	C*8	Blanks		Names of fields to output as 1-dimensional to history file.
FLXAVE	L	.TRUE.		If true, only communicate with the flux coupler on time steps where albedos are calculated.
HYDRO	L	.TRUE.		Used prognostic hydrology if true.
LOCPNH	C*80	'.'		Directory where output history files are written.
LOCPNR	C*80	'.'		Directory where output restart files are written.
LSMGEO	I	23		Output grid type in "mn" format

				where: m = 1: regular grid m = 2: gaussian grid n = 1: grid starts at dateline; western edge ON dateline n = 2: grid starts at Greenwich; western edge ON Greenwich n = 3: grid starts at Greenwich; centered ON Greenwich Only used when FINIDAT is set to "arbitrary initialization"
NREVSN	C*80	blanks		Full restart pathname for branch run.
NSWRPS	C*8	blanks		Mass store, write password, for output files.
PERGRO	L	.FALSE.		Flag for random perturbation test. Set TRUE for numerics testing purposes only. Otherwise leave FALSE.
REST_PFILE	C*80	'~/lnd.CASEID.rpointer'		Full UNIX pathname of restart pointer file.

The model is capable of executing using default values for most of the namelist variables (most default values are only appropriate at the T42/18-level resolution). However, some of the namelist variables must be explicitly set by the user in order for the model to run. CCMEXP variables that **must** be set are CASEID, NESTEP (or NELAPSE), NCDATA, BNDTVS, BNDTVO, and some combination of the orbital parameters (IYEAR_AD, or MVELP, ECCEN, and OBLIQ) as discussed at the bottom of Table 2.5. The only required LSMEXP variable is FINIDAT.

2.3 Model Input Datasets

CCM3.6 is a combination of atmosphere, land, and ocean components. Input datasets needed for the atmospheric component define initial state and ozone boundary data. The land component requires initial state data over land only. If the ocean component is run as a slab ocean model (SOM) it requires initial state and boundary data, whereas if it is run as a data ocean model (DOM), it requires SST input data. The default ocean configuration is the data ocean model. All initial state and boundary data are in netCDF format. All time-varying input datasets (e.g. ozone boundary dataset) are based on 365 days per year with no leap years.

Initial and ozone boundary data for the atmosphere component is specified via the CCMEXP namelist variables NCDATA and BNDTVO, respectively. The ocean boundary dataset is specified via the CCMEXP namelist variable BNDTVS. Finally, the initial land dataset is obtained from the LSMEXP namelist variable FINIDAT. Initial state data are only provided at T42 resolution.

See "Obtaining the Source Code and Datasets" for information describing how to obtain these datasets.

2.3.1 Atmosphere Component Datasets

The atmospheric component requires initial conditions and ozone datasets.

2.3.1.1 Initial Conditions Dataset

The, initial conditions, dataset is specified by namelist variable NCDATA. This dataset contains initial values of prognostic variables (U,V,T,Q,PS, and TS1 through TS4), the surface geopotential field (PHIS), the orography flag (ORO), and the standard deviation of geopotential height (SGH). Required initial fields are outlined in the table below. By default, the model writes an instantaneous history file containing these fields each simulated month (controlled by namelist variable INITHIST) for possible use as an initial dataset in another run. In order to use one of these files as an initial dataset, the utility **ccm2nc** must first be run to convert it to netCDF format.

Table 2.7: Atmospheric Component Initial Dataset Fields

History file Field Name	Variable Name	Description
Multi-Level Fields		
T	T	temperature (K)
U	u	zonal wind component (m/sec)
V	v	meridional wind component (m/sec)
Q	q	water vapor specific humidity (K_{gH2O}/K_{gair})
Single Level Fields		
PHIS	Φ_s	surface geopotential (m^2/s^2)
PS	P_s	surface pressure (Pa)
SGH	SGH	standard deviation of geopotential height (m)

`ORO`	ORO	surface type flag: = 0 over ocean = 1 over land = 2 over sea ice
`TS1,TS2,TS3,TS4`	T_s	Four subsurface temperature levels `(K)` (`TS1` is surface temperature)

All fields are instantaneous values. In addition to the prognostic fields, the initial dataset contains information on the time for which the data are valid, the dimensionality of the fields, the spectral truncation, and the latitudes, longitudes, and vertical levels of the data.

When running with the SOM, the standard initial dataset is replaced by a SOM initial dataset. In addition to the initial fields shown above, the SOM initial dataset requires a sea ice thickness (`SICTHK`) and snow cover (`SNOWH`) to initialize the ice component of SOM as shown below.

Table 2.8: Additional Slab Ocean Model Initial Dataset Fields

History file Field Name	Variable Name	Description
`Single Level Fields`		
`SICTHK`	h_i	**sea ice thickness (m)**
`SNOWH`	h_s	**Snow cover (liquid water equivalent) (m)**

An 18-level initial dataset on a, 128 X 64, transform grid (T42) is provided with the CCM3.6 dataset tar file.

2.3.1.2 Producing Initial Condition Datasets at Various Resolutions

In order to run the model at other horizontal or vertical resolutions, utility programs have been included to create an, initial conditions, file at whatever resolution the user wishes. Directory **ccm/tools/interpic** (created when untarring ccm3.6.code.tar) contains code to interpolate a netCDF format CCM3 history file to a 3-D grid specified by command-line arguments. If a horizontal interpolation is performed (by specification of **-x** and/or **-y** command line arguments) the required input field ORO **will not be produced** on the output file. Otherwise, ORO will be produced. ORO is a land/ocean/sea-ice discrete flag field for which horizontal interpolation would be meaningless. For the case where ORO is not produced, the user must then refer to directory **ccm/tools/definesurf .** This directory contains code to read a high-resolution topography dataset and produce the required input fields SGH, PHIS, and ORO on an existing CCM3 history file in netCDF format. The high resolution topography dataset is named `topo.nc` and is included with the standard CCM datasets tar file. The user is referred to README files in the above directories for further information and specific instructions on how to build and run these utilities. An example sequence of commands to produce a horizontally interpolated T21 initial dataset would be:

interpic -x 64 -y 32 SEP1.T42.0198.nc SEP1.T21.nc
definesurf -t topo.nc SEP1.T21.nc

Because of the **horizontal interpolation** from T42 to T21, **both** of the above commands are required and must be run in the indicated sequence for a complete initial dataset to be produced. **interpic** interpolates the initial prognostic fields and **definesurf** creates the T21 ORO field from the high resolution topography.

2.3.1.3 Ozone Dataset

The ozone boundary dataset contains ozone volume mixing ratios. These values are interpolated to the model vertical levels at each model grid point. The ozone dataset is in netCDF format and contains the fields `PS` and `OZONE` . `PS` is a constant 1000 mb pressure field. The ozone dataset is specified by namelist variable `BNDTVO`.

2.3.2 Ocean Component Datasets

The ocean component can be run in one of two ways, as a data ocean model, which reads input sea surface temperature data, or as a slab ocean model, which computes SST's.

2.3.2.1 Running With the Data Ocean Component

Running the data ocean component requires a time-variant SST dataset. The standard SST dataset contains 12 monthly time samples. The model can also read multi-year SST datasets. The SST dataset is specified by namelist variable `BNDTVS`.

A SST dataset on a, 128 X 64, transform grid (T42) is provided with the CCM3.6 dataset tar file. This dataset is named T42M5079.nc

In order to run the model at other horizontal resolutions, a utility program has been included to create a SST dataset at whatever resolution the user wishes. Directory `ccm/tools/interpsst` (created when untarring ccm3.6.code.tar) contains code to interpolate a netCDF format SST file to any gaussian 2-D grid specified by command-line arguments. The user is referred to `README` files in the above directory for further information and specific instructions on how to build and run these utilities. An example sequence of commands to produce a horizontally interpolated T21 SST dataset would be:

interpsst -x 64 -y 32 T42M5079.nc sst.T21.nc

In addition to the standard T42 resolution SST file, the Shea, Trenberth, Reynold (NMC/CAC) dataset STRsst2x2.nc has also been provided in the ccm/tools/interpsst directory. This dataset uses the Reynolds SST climatology (the COADS/ICE climatology) and includes improvements in the regions of the Gulf Stream and Kurishio Current, i.e. reinstated more realistic horizontal temperature gradients. (The T42 SST dataset provided with the distribution was derived from the Shea, Trenberth, Reynold dataset.)

2.3.2.2 Running with the Slab Ocean Model Component

This section documents an unsupported feature of the current distribution. Code and full documentation will be provided in the future.

To configure the model to run with a slab ocean model requires the setting of the cpp token `COUP_SOM` in the misc.h file. The SOM input fields ocean mixed layer depth and ocean mixed layer heat flux are shown below. These fields replace the sea surface temperature field of the standard SST dataset. Mixed layer depths are specified from observations and never change from one SOM run to the next. Mixed layer heat flux depends upon a control run, and thus can vary from one SOM run to the next (the user would need to calculate them unless the standard control is used). Ocean mixed layer heat flux simulates deep water heat exchange and ocean transport. The mixed layer depth and heat flux evaluation is described fully in the model description (Kiehl et. al., 1996).

Table 2.9: Slab Ocean Model Time-Varying Boundary Dataset Fields

History file Field Name	Variable Name	Description
Single Level Fields		
`QO`	Q	Ocean mixed layer heat flux (W/m^2) (12 monthly averages)
`MLD`	h_o	mixed layer depth (m) (12 monthly averages)

2.3.3 Land Surface Model Input Datasets

The initial LSM dataset contains time-invariant and time-variant data (see "LSM initial runs "). Time-invariant data fields provide information about each grid cell on the two-dimensional LSM grid and are described in the following table:

Table 2.10: Land Surface Model: Initial Dataset Time Invariant Fields

Field Name	Description
`latixy`	latitude

longxy	longitude
surf2d	surface type
soic2d	soil color
sand2d	percent sand
silt2d	percent silt
clay2d	percent clay
pctlak	percent lake
pctwet	percent wetland
drngarea	LSM grid cell areas
drngfrac	Fractional drainage matrix
drngbasn	Basin type drainage matrix

Time variant initial data fields are 1-dimensional vectors for the LSM subgrid points. The fields are summarized in the following table and are all single level except where indicated.

Table 2.11: Land Surface Model: Initial Dataset Time Variant Fields

Field Name	Description
h2osno	Snow water (K_{gH20}/m^2)
h2ocan	Canopy water (K_{gH20}/m^2)
h2osoi	Volumetric soil water content (mm^3_{H20})/(mm^3_{soil}) (multi-level) (0<=h2osoi<=watsat)
tv	Vegetation temperature (K)
tg	Ground temperature (K)
tsoi	Soil temperature (K) (multi-level)

Initial LSM data can be generated at run time or can be obtained from a previously generated LSM initial dataset (see Initial runs in the LSM User's Guide). If LSM namelist variable FINIDAT is set to the name (or full pathname) of an LSM initial file, initial LSM data will be obtained from that file. The LSM initial dataset **must**, however, have the same grid resolution and number of LSM subgrid points as the run for which it is intended.

Alternatively, if `FINIDAT` is set to `"arbitrary initialization"`, LSM initial data fields are computed at run time and output to file `lsmi_arbini.nc`. In this case, LSM initializes the time-variant initial fields to reasonable values (but which are not in equilibrium with the atmosphere). Six additional datasets independent of the LSM grid resolution are also required. These files are summarized below:

Table 2.12: Land Surface Model: Resolution Independent Datasets

Dataset	Description
olson.data	land type data on 1/2 x 1/2 degree grid (ASCII format)
bats.data	soil color data on T42 gaussian grid (ASCII format)
webb.1x1	soil texture data on 1x1 degree grid (ASCII format)
flak.1x1	inland water data on 1x1 degree grid (ASCII format)
swmp.1x1	swamp data on 1x1 degree grid (ASCII format)
drainage.p5xp5.nc	drainage data on 1/2x1/2 degree grid (netCDF format)

LSM initial datasets are simply instantaneous LSM netCDF history files containing the fields necessary for initialization. During a model run, they can be generated monthly, yearly or not at all depending on the value of the LSM namelist variable `INITHIST`. LSM initial datasets produced during a model run have the names `lsmi_yyyymmdd_sssss.nc` where `yyyy` is the year, `mm` is the month, `dd` is the day, and `sssss` is the seconds into the day of the simulation.

When the LSM attempts to acquire input data files, it first looks in the current working directory (i.e. the model executable directory). If the required files are not there, the directory specified by the LSM namelist variable, `DATADIR`, is searched. If the files are still not found, it looks for the full pathname of the file, first on disk and then on the MSS (Mass Storage System). Note that if the LSM initial data is created at run time, the six additional input datasets **must** either be in the model executable directory or in the directory specified by `DATADIR`.

2.4 Example Namelists

In the following, several example model runs are presented with their associated namelist input. These examples were chosen to illustrate most of the namelist variables described in Sec. 2.2 . It is assumed in these examples that all required input datasets are located in the directory where the model executable resides. It is also assumed that the data ocean model is used.

2.4.1 Example 1 -- Initial Run, One Day

This is a one-day initial run beginning September 1. Restart files will be archived with a retention period of 100 days and all history files will be packed 2:1. The use of the NCAR packing function and the NCAR MSS make this namelist suitable for an NCAR machine only. All required datasets are included as part of the CCM3 release.

```
&CCMEXP
CASEID = 'run01'
CTITLE = 'Test run: climatological SST'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSWRPS = 'passwd'
IRT    =    100
NSREST =      0
NDENS  =      2
NNBDAT = 00000901
NNBSEC =      0
NNDBAS =      0
NNSBAS =      0
MFILT  =     20
DTIME  =  1200.
NESTEP =     72
NHTFRQ =     72
INCORBUF = .false.
INCORHST = .false.
INCORRAD = .false.
IYEAR_AD = 1950
/

&LSMEXP
finidat = 'arbitrary initialization'
/
```

CASEID

The case identifier for this run is `run01`. It is recorded in the history file header record, the model printed output, and is used as part of the MSS path names for the history and restart files. `CASEID` must be 8 (or less) valid UNIX filename characters.

CTITLE

This title will appear in the model output and in the history file headers.

NCDATA

The, initial conditions, dataset is a netCDF file called `SEP1.T42.0198.` See "Model Input

Datasets" .

BNDTVS

Specifies the time-variant sea surface temperature (SST) dataset. This file must be in netCDF format.

BNDTVO

Specifies the time-variant ozone dataset `ozn.0596.r8.nc`. This dataset is defined on a pressure grid (unlike the hybrid grid of the model) and is in netCDF format.

NSWRPS

Mass Store, write password, for all output datasets from this run. This password must remain the same throughout a case.

IRT

Retention period of files written to the NCAR Mass Storage System. History files from this run will be written with a retention period of 100 days.

NSREST

Setting `NSREST = 0` requests an initial run starting from the initial dataset, `NCDATA`. Continuation (restart and branch) runs may be specified via this variable. See Examples 3 and 5 for more information on continuation runs.

NDENS

Specifies the packing factor used for model history files. Data packing is available only on an NCAR Cray and results in a loss of precision in the data values. In this example, the history files are packed 2 to 1.

NNBDAT, NNBSEC, NNDBAS, NNSBAS

These time variables allow the user to specify the base date and seconds as well as the base day (starting day number of run). If these variables are not defined the model will use date information from the initial dataset. In this example, `NNBDAT` sets the base date for this run to "`00000901`", or September 1, year 0000.

MFILT

Specifies the number of time samples to be written to each history file. The model will open a history tape and write `mfilt` time samples before closing the file. Setting `MFILT` to 20 means that each complete output history file will contain 20 time samples.

DTIME

This specifies the timestep for the run in seconds (floating point). In general changes in resolution require a change in `DTIME`.

NESTEP

Specifies the ending timestep for the model run. This example will end at timestep 72. Positive integer values denote timesteps while negative integer values denote model days. The same ending time could have been specified as `NESTEP=-1`.

Hint: When starting a long run, it is a good idea to make a 1-day initial run, then restart the case for the remainder of the experiment. If the system crashes and the job is rerun, the model will restart from the last restart files. If on the other hand, you specified your model run as one long, initial run, the job would reload the initial namelist and start again from the beginning, duplicating all model results from the aborted job.

NHTFRQ

Specifies the number of timesteps (if positive) or the number of model hours (if negative) between history file writes. To write a file every day would require running 72 timesteps between model writes (72 *1200 seconds/step = 86400 seconds or 1 day). Alternatively you could set `NHTFRQ` to -24 to get the same output frequency. Timestep 0 data will always be written to the history file. `MFILT` set to 20 means that each history file will contain 20 time slices of data.

INCORBUF, INCORHST, INCORRAD

Specifies whether model buffers are stored in-core or out-of-core (written to disk). The target machine for this run is a Cray with SSD. Availability of the high speed SSD allows CCM3.6 to run out-of-core with minimal overhead. When running on a machine with no external high-speed I/O device (SSD), namelist variables `INCORBUF`, `INCORHST`, and `INCORRAD` should be `.true.` (Default).

IYEAR_AD

Calculate the earth's orbital parameters for 1950 AD. See Example 4 or Table 2.5 for more information on how to set variable. The year 1950 is used to most closely match the orbital parameters of previous releases of CCM3.

FINIDAT

LSM initial dataset. "arbitrary initialization" requests that the LSM use hard coded values for initial state information instead of reading the data from an input file.

2.4.2 Example 2 -- Initial Run, T21

Input variables shown in this example request an "initial" run. The model has been compiled to run at T21 resolution using input datasets created with the interpolation tools provided with the distribution. History files are not packed and no files will be archived. This namelist has no NCAR specific functionality and is suitable for running on non-NCAR machines.

```
&CCMEXP
CASEID = 'run02'
CTITLE = 'Test run'
NCDATA = 'SEP1.T21.nc'
BNDTVS = 'SST.T21.nc'
BNDTVO = 'ozn.0596.r8.nc'
IRT    =      0
DTIME  =  1800.
NSREST =      0
NDENS  =      1
NNBDAT = 000901
NNBSEC =      0
NNDBAS =      0
NNSBAS =      0
MFILT  =     20
NESTEP =     72
NHTFRQ =     72
IYEAR_AD =  1950
INCORBUF = .true.
INCORHST = .true.
INCORRAD = .true.
/
&LSMEXP
finidat = 'arbitrary initialization'
/
```

IRT

No history or restart files will be archived. If there is no connection to an archival device, `IRT` should be set to zero and output datasets will be written to disk. If this variable is non-zero, the model will build an archival command that it will execute as a shell command.

NCDATA

The initial dataset `SEP1.T21.nc` is interpolated from the T42 resolution dataset provided with the distribution by the utilities, **interpic** and **definesurf.** **interpic** interpolates a CCM history file in netCDF format to a user-specified resolution. **definesurf** sets required CCM surface fields SGH, PHIS, and ORO based upon a high-resolution dataset (file `topo.nc` provided with the distribution model datasets). An example sequence of commands to produce `SEP1.T21.nc` would be:

interpic -v -x64 -y32 SEP1.T42.0198.nc SEP1.T21.nc

definesurf -v -t topo.nc SEP1.T21.nc

See the section on Sec. 2.3.1 "Atmospheric Component Datasets" for more information regarding these utilities.

BNDTVO

The default T42 ozone dataset can be used with any resolution run. The model will interpolate the dataset to the correct resolution.

BNDTVS

The time varying sea surface temperature (SST) dataset `SST.T21.nc` is created using the utility **interpsst** provided with the distribution. **interpsst** interpolates the 2X2 degree SST dataset (`STRsst2x2.nc`) provided with the utility. An example command to produce `SST.T21.nc` would be:

interpsst -v -x64 -y32 STRsst2x2.nc SST.T21.nc

See the section on Sec. 2.3.2 "Ocean Component Datasets" for more information regarding this utility.

NDENS

No packing will be performed. Packing can only be enabled (i.e. set to a value greater than 1) on an NCAR Cray.

DTIME

1800 seconds is an appropriate timestep for T21 resolution.

Resolution-dependent namelist variables that should be changed are: `DTIME` , `DIF2` , and `DIF4`.

2.4.3 Example 3 -- Restart Run, 30 Days

The following example restarts the run given in Example 1 and simulates another 30 days. Since `NHTFRQ=72`, a time sample of data will be written to the history file once per day (with a timestep of 1200 seconds, 72 timesteps equals one day). Also, because the default value of `NINAVG` is used (i.e. `NINAVG(1)='A'`), each variable in the time sample will be averaged over the previous 72 timesteps (except `ORO` and `PHIS` which are always instantaneous).

NOTE: in a restart run, aside from changing `NSREST` and `NESTEP` or `NELAPSE`, the exact same namelist should be submitted as was used in the initial run.

```
&CCMEXP
CASEID = 'run01'
CTITLE = 'Test run: climatological SST'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSWRPS = 'passwd'
IRT    =    100
NSREST =      1
NDENS  =      2
NNBDAT = 000901
NNBSEC =      0
NNDBAS =      0
NNSBAS =      0
MFILT  =     20
DTIME  =   1200.
NELAPSE =   -30
NHTFRQ =     72
INCORBUF = .false.
INCORHST = .false.
INCORRAD = .false.
IYEAR_AD = 1950
/
&LSMEXP
finidat = 'arbitrary initialization'
/
```

CASEID

CASEID must be the same as the case, which is being continued.

NSREST

NSREST=1 indicates a "restart" run, which starts from the most recent restart datasets and continues the case.

NELAPSE

This run, as a continuation of the previous one-day simulation, will start at the beginning of day 2 and run for 30 days, ending after day 31, i.e., `NELAPSE=-30` means "run 30 days beyond the end of the previous run." This could also be specified by setting `NESTEP=-31`(days) or `NESTEP=2232` (timesteps), assuming a `DTIME` of 1200 seconds.

2.4.4 Example 4 -- Paleo run

This is an initial run for a period of one day beginning on September 1, 6000 BC. This namelist defines the earth's orbit, solar constant, and the volume mixing-ratio for the greenhouse gases. Normally these values are changed for a paleo-climate simulation.

```
&CCMEXP
CASEID = 'run04'
CTITLE = 'Paleo-climate run: 6000 BC'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
IRT    =      0
DTIME  =  1200.
NSREST =      0
NDENS  =      1
NNBDAT = 000901
NNBSEC =      0
NNDBAS =      0
NNSBAS =      0
MFILT  =     20
NESTEP =     72
NHTFRQ =     72
INCORBUF = .true.
INCORHST = .true.
INCORRAD = .true.
ECCEN = 0.01868126
OBLIQ = 24.10538
MVELP = 0.87
CO2VMR = 280.
CH4VMR = 700.
N2OVMR = 275.
F11VMR = 0.
F12VMR = 0.
SCON = 1.363e6
/
&LSMEXP
finidat = 'arbitrary initialization'
/
```

`ECCEN, OBLIQ, MVELP`

Set the earth's orbital parameters for year 6000 BC. For the most accuracy, the parameters `ECCEN, OBLIQ`, and `MVELP IYEAR_AD`, should be set, rather than calculating values by using `IYEAR_AD`.   When `IYEAR_AD` is specified, the other orbital namelist parameters are ignored. If you want to calculate the orbital parameters for 6000 BC using (Berger, 1977) replace the settings for `ECCEN, OBLIQ`, and `MVELP` with.

```
 IYEAR_AD = -6000
```

CO2VMR,CH4VMR,N2OVMR,F11VMR,F12VMR

Set the volume mixing ratios for the greenhouse gases: CO_2, CH_4, N_2O, CFC_{11}, and CFC_{12}. The values are set to values that might be reasonable for 6000 BC.

SCON

Set the solar constant to 1363 W/m^2.

BNDTVO,BNDTVS

In this example these datasets would not be appropriate for a paleo run. In general, you will want to create your own datasets to provide these values.

2.4.5 Example 5 -- Branch Run

In the following example, a new case is begun by "branching" from a CCM3 control run. The branch run allows the user to change any namelist variables associated with the history files, while continuing from a chosen point in a previous run. Other namelist variables should not be modified. A branch run must always start a new case, so the namelist variable `CASEID` must be changed. In this example, some fields are excluded from the primary history file, and placed on two auxiliary history files, to be written every 18 timesteps. Note the use of `NSREST=3` for a branch run, and the additional values assigned to `NDENS`, `MFILT`, and `NHTFRQ` for the auxiliary file.

```
&CCMEXP
CASEID = 'run05'
CTITLE = 'CCM3 branch run'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSREST = 3
NSWRPS = 'mypass'
IRT    = 365
RIRT   = 365
NREVSN = '/USERNAME/run01/ccm3/rest/ccmr_00000902_00000'
NHTFRQ = 72,18,18
NDENS  = 2,2,2
MFILT  = 10,30,30
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
NELAPSE = -30
NINAVG = 'A','X','M'
AUXF   = '1','PRECL','PRECC','CLOUD',
         '2','PRECL','PRECC','CLOUD'
EXCLUDE = 'PRECL','PRECC','CLOUD'
IYEAR_AD = 1950
/

&LSMEXP
nrevsn = 'lsmr_00000902_00000'
/
```

CASEID

A branch run must always start a new case.

NSREST

NSREST=3 indicates a "branch" run, which starts from a specific set of restart datasets and creates a new case.

NREVSN

This run uses master restart file `ccmr_00000902_00000` from example 1 to provide the initial dataset for starting this case. *USERNAME* in this example refers to the users logid. A branch run also requires

that `NREVSN` be set for the lsm, again we use the lsm restart file written by example 1.

NHTFRQ

History file write frequency will be every 72 timesteps (1 day) for the primary file, and every 18 timesteps (6 hours) for the two auxiliary files.

NDENS

All primary and auxiliary history files will be packed 2 to 1.

MFILT

10 time samples will be written to each primary history file. 30 time samples will be written to each auxiliary history file.

NINAVG

NINAVG = ’A’,’X’,’M’ specifies the field type in each history file. The primary history file will be averaged (daily averaged since `NHTFRQ(1)=72`). Fields on the first auxiliary history file will contain the maximum values of the `PRECL, PRECC` and `CLOUD` fields for the past 6 hours (18 timesteps), and fields on the second auxiliary history file will contain the minimum values of the same fields for the past 6 hours.

AUXF

This list specifies that two auxiliary history files will be written containing the fields shown. Up to five such files may be declared.

EXCLUDE

To save file storage space, the fields that will be written to the auxiliary history files above are excluded from the primary history file.

2.4.6 Example 6 -- Initial Run, with additional fields

The following namelist specifies an initial run with changes to the default primary history file. Changes include adding certain fields that are in the Master Field List (see Table 2.13) to the primary history file, and making all of the values in the file instantaneous. Other optional changes are described below.

```
&CCMEXP
CASEID = 'run06'
CTITLE = 'CCM3 standard run'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSREST = 0
NSWRPS = 'mypass'
IRT    = 1825
RIRT   = 365
NDENS  = 2
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
MFILT  = 10
NELAPSE = -30
NHTFRQ  = 72
NINAVG  = 'I'
PRIMARY = 'PRECSL','PRECSC'
IYEAR_AD = 1950
/

&LSMEXP
finidat = 'arbitrary initialization'
/
```

IRT

History files will be archived with a retention period of 5 years.

RIRT

Restart datasets will be archived with a retention time of 365 days.

NINAVG

All fields on the history file will be instantaneous values.

PRIMARY

This list defines additional fields on the primary history file. These fields must be in the

Master Field List, and the appropriate calls to subroutine `outfld` must be uncommented. See "Adding New Variables" for more information.

2.4.7 Example 7 -- Run with Multiyear SST Dataset

This is a 10-year run using a multi-year SST dataset. The base date and time information (`NNBDAT` and `NNBSEC`) for the model run must fall within the time frame covered by the SST dataset. In this case, the namelist variable `SSTCYC` is set to `false` in order to retrieve SST data for the proper year and month of the model run. When `SSTCYC` is set to `true` the model ignores year information and only cycles over the first twelve months of data.

```
&CCMEXP
CASEID = 'run07'
CTITLE = 'multiyear SST (AMIP)dataset'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'sst_jan1978-sep1993.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSREST = 0
NSWRPS = 'mypass'
IRT    = 3650
RIRT   = 365
NDENS  = 2
NNBDAT = 19790901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
SSTCYC = .false.
MFILT  = 10
NELAPSE = -3650
NHTFRQ = 72
IYEAR_AD = 1950
/

&LSMEXP
finidat = 'CCM3.3LSMICSEP1_with_drng.nc'
/
```

BNDTVS

This dataset must contain multiyear SST data for the time period over which the model will be integrated. In this example the base date of the run is determined by `NNBDAT`. If an end-of-file is read on the SST dataset, the code will abort rather than rewinding to recycle the data as in a single-year dataset.

NNBDAT, NNBSEC

These variables set the base date and seconds of the model run to `SEP 1, 1979` to be commensurate with dates on the SST dataset. If not input, the base date from the initial dataset will be used as the simulation base date.

SSTCYC

Setting `SSTCYC` to `.false`. indicates that a multiyear SST dataset will be input.

FINIDAT

This is the T42 LSM initial dataset for SEP 1. Again the SEP 1 dataset is used to correspond with the CCM model start date.

2.4.8 Example 8 -- Generate a Monthly-Averaged History File

In this example, CCM3 is run for two months, generating monthly-averaged history files at the end of each month. Output history filenames for this run will be of the form `yyyy-mm`, where `mm` is the month and `yyyy` is the year for which the data are valid. The first file output for September of year 0 will be `0000-09`. Since the primary history file is monthly-averaged, the values of `NHTFRQ(1)` and `NREFRQ` are ignored.

```
&CCMEXP
CASEID = 'run08'
CTITLE = 'Monthly average run'
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSREST = 0
NSWRPS = 'mypass'
IRT    = 365
RIRT   = 365
NDENS  = 1,2
MFILT  = 2,2
AUXF  = '1','T','U'
NINAVG = 'Q','A'
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
NELAPSE= -61
NHTFRQ = 72,72
IYEAR_AD = 1950
/
&LSMEXP
finidat = 'arbitrary initialization'
/
```

AUXF

This run defines one auxiliary history file.

NINAVG

A monthly averaged primary history file is requested by setting the first value of `NINAVG` to '`Q`' while the auxiliary history file data are daily averaged.

2.4.9 Example 9 -- Debugging Run

While the primary history file is not written to the Mass Store, the disk copy will be left behind (not removed) for fast access and analysis. Only 10 timesteps will be run, with history data being written every timestep. Output will be line-buffered to ensure printout will occur each timestep. For efficiency, no restart data will be written.

```
&CCMEXP
CASEID = 'examp09'
CTITLE = 'Debugging run'
LINEBUF = .TRUE.
NCDATA = 'SEP1.T42.0198.nc'
BNDTVS = 'T42M5079.nc'
BNDTVO = 'ozn.0596.r8.nc'
NSREST = 0
NREFRQ = 0
IRT    = 0
NDENS  = 1
MFILT  = 10
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
NELAPSE= 10
NHTFRQ = 1
IYEAR_AD = 1950
/
&LSMEXP
finidat = 'arbitrary initialization'
/
```

IRT

When `IRT=0`, the output history file is only written to the local disk and not archived.

NDENS

Data are unpacked to maintain full accuracy.

MFILT

10 time samples of unpacked data will be written to each history file.

LINEBUF

Printout will be flushed with line generated.

NREFRQ

Value of zero prevents restart files from being written.

NHTFRQ

History data are written every timestep.

2.5 Model Output Datasets

CCM3 produces a series of binary history files containing the atmospheric gridpoint data generated during the course of a run and a series of restart files necessary to continue the run. These will be described below.

2.5.1 Model History Files

History files contain model data values written at specified times during a run. The user can specify the frequency at which the data are written. Options are also available to record averaged, instantaneous, maximum, or minimum field values. Namelist options to control the frequency and type of output apply to all fields on a history file. They will all be output at the same frequency (i.e. daily) and will be of the same type (i.e. averaged; except for `PHIS` and `ORO` which are always instantaneous). If the user wishes to see a field written at different time frequencies (e.g. daily, hourly) additional history files must be declared.

Two kinds of history files can be output, "primary" or "auxiliary". A primary history file is always output, whereas auxiliary files are optional. The exact contents of the primary file may be customized using namelist variables `PRIMARY` and `EXCLUDE`. Auxiliary history files are created only if requested by the user via namelist variable `AUXF`. Up to five auxiliary history files can be declared if namelist variable `INITHIST` is set to `'NONE'` and four otherwise.

History files may be examined by various postprocessors. One example is the CCM Modular Processor, which still exists but is no longer maintained. Alternatively, "**ccm2nc**", a tool to convert ccm-history files to netCDF format, is now supported. This allows a user to employ any of a number of available tools (both commercial and share-ware) to visualize or otherwise analyze their output history files. Examples include the "CSM Processor" NCAR Graphics package, FERRET, ncview, MATLAB, AVS, IDL, and Yorick. For a list of software tools for interacting with netCDF files view the UNIDATA netCDF web-site: http://www.unidata.ucar.edu/packages/netcdf/software.html.

Values recorded for fields on a history file can be represented in one of four different ways. Data may be time averaged since the last write to the history file, instantaneous, or point-by-point maximum or minimum over the time interval. Namelist variable `NINAVG(i)` controls this function (index "i" here refers to the file index, 1 means the primary file, 2 means 1st auxiliary, etc.). Possible values are 'A' - time averaged, 'I' - instantaneous, 'M' - point minimum, 'X' - point maximum, and 'Q' - monthly averaged ('Q' is allowed only to apply to the primary history file). The default value for each history file is 'A'. Fields in a history file are each of the type indicated by `NINAVG(i)`, except `ORO` and `PHIS` which are always instantaneous.

The time frequency at which data are written to the history file is specified by `NHTFRQ(i)`. As an example, consider the namelist input shown in Example 5. Here, `NHTFRQ(1)=72` and `NINAVG` is set to 'A'. Every 72 timesteps a time sample of data is written to the primary history file, with each field being time averaged over the previous 72 timesteps.

The number of time samples written to a single history file, primary or auxiliary, is set via namelist variable `MFILT(i)`. The packing density of fields in each history file is set with namelist variable `NDENS(i)` (only available on an NCAR Cray). This namelist variable should only be 1, 2, or 4. A packing density of 4 to 1 `(NDENS(i)=4)` could result in significant loss of precision.

Refer to Table 2.5 and Table 2.6 in Sec. 2.2 for a complete description of all history file namelist variables.

2.5.1.1 Primary History File

The following table contains a list of fields, referred to as the "Master Field List", that can currently be written to the primary or auxiliary history files. The first column indicates whether the field will automatically be included on the primary history file. Fields not on the default primary history file may be included via namelist variable `PRIMARY` (see "Example 6 -- Initial Run, with additional fields "). Conversely, any fields included by default may be removed using namelist variable `EXCLUDE` (refer to "Example 5 -- Branch Run"). Some field names appearing in the second column contain two upper case letters followed by two lower case `x`'s (e.g. `TRxx`, `DCxx`, etc.). These fields correspond to passive tracer constituents that may be transported. In the history file, the two `x`'s will be replaced by a two-digit tracer number (refer to "Adding New Constituents). The third column, associates the history file field name with its mathematical symbol as given in *Description of the NCAR Community Climate Model (CCM3)* (Kiehl *et al.*, 1996). The fourth column provides a brief description of the field. In the column labeled "NL" a "1" indicates a single-level field and an "N" indicates a multilevel field (on `plev` vertical levels). The last column in the table shows the physical units associated with each field.

Table 2.13: Master Field List

Default File?	Field Name	Symbol	Field Description	NL	Units
yes	`Q`	q	specific humidity	N	Kg/Kg_{air}
yes	`TRxx`	q	advected tracers other than specific humidity	N	Kg/Kg_{air}
yes	`NAxx`	q	non-advected tracers	N	Kg/Kg_{air}
yes	`T`	T	Temperature	N	K
yes	`U`	u	zonal wind component	N	m/s
yes	`V`	v	meridional wind component	N	m/s

yes	PS	P_s	surface pressure	1	Pa
yes	TS	T	surface temperature	N	K
yes	TS1	T_s	ice temperature (level 1)	1	K
yes	TS2	T_2	ice temperature (level 2)	1	K
yes	TS3	T_3	ice temperature (level 3)	1	K
yes	TS4	T_4	ice temperature (level 4)	1	K
yes	PHIS	ϕ_s	surface geopotential	1	m^2/s^2
yes	DCxx	$\left(\frac{\delta q}{\delta t}\right)_A$	tracer tendency from adjustment physics	N	$(Kg/Kg_{air})/s$
yes	TAxx	$\frac{dq}{dt}$	total advection tendency of tracer	N	$(Kg/Kg_{air})/s$
yes	VDxx		vertical diffusion tendency of tracer	N	$(Kg/Kg_{air})/s$
no	HAxx		horizontal advection tendency of tracer	N	$(Kg/Kg_{air})/s$
no	VAxx		vertical advection tendency of tracer	N	$(Kg/Kg_{air})/s$
no	DFxx		SLT fixer tendency of tracer	N	$(Kg/Kg_{air})/s$
no	TExx		time tendency of tracer	N	$(Kg/Kg_{air})/s$
no	SSxx		tracer source/sinks (pcnst-1 values)	N	$(Kg/Kg_{air})/s$
no	SFxx		tracer surface fluxes (pcnst-1 values)	N	$(Kg/m^2)/s$
no	DUH		*u* horizontal diffusive heating rate	N	K/s
no	DVH		*v* horizontal diffusive heating rate	N	K/s

yes	DTH	$\hat{F}_{TS}\left(T^{*}\right)$	*T* horizontal diffusion	N	K/s
yes	ORO		surface type flag: = 0 for ocean = 1 for land = 2 for sea ice	1	flag
yes	SNOWH	S_n	water equivalent snow depth	1	m
yes	TREFHT		Surface dependent reference height temperature	1	K
yes	LWSH		Liquid water scale height	1	m
yes	ETADOT	$\dot{\eta}$	Vertical motion on half levels	N	1/s
yes	PRECL	P_{ls}	large-scale stable precipitation	1	m/s
yes	PRECC	P_{cs}	convective precipitation	1	m/s
no	PRECSL		large-scale stable snowfall	1	m/s
no	PRECSC		convective snowfall	1	m/s
yes	SHFLX	$c_p \rho_1 \left(\overline{w'\theta'}\right)_o$	surface sensible heat flux	1	W/m^2
yes	LHFLX	$L \rho_1 \left(\overline{w'q'}\right)_o$	surface latent heat flux	1	W/m^2
yes	QFLX	$\rho_1 \left(\overline{w'q'}\right)_o$	surface water flux	1	$(Kg/m^2)/s$
yes	PBLH		height of planetary boundary layer	1	m
yes	USTAR	u^*	surface friction velocity	1	m/s
no	CGH		pbl nonlocal transport, heat	N	K/m
no	CGQ		pbl nonlocal transport, humidity	N	1/m

yes	CGS		Counter-gradient coefficient on surface kinematic fluxes	N	s/m^2
yes	TPERT	θ'_0	pbl plume temperature perturbation	1	K
yes	QPERT	q'_0	pbl plume moisture perturbation	1	Kg/Kg
yes	ZMDQ		Q tendency - Zhang moist convection	1	$(Kg/Kg_{air})/s$
yes	ZMDT		T tendency - Zhang moist convection	1	$(Kg/Kg_{air})/s$
yes	KVH		diffusivity for heat	N	m^2/s
no	KVM		diffusivity for momentum	N	m^2/s
no	DUV		*u* vertical diffusion	N	m/s^2
no	DVV		*v* vertical diffusion	N	m/s^2
yes	DTV		*T* vertical diffusion tendency	N	K/s
yes	FSNS	$F^S{}_N(P_s)$	net downward solar flux at surface	1	W/m^2
yes	FLNS	$F^L{}_N(P_s)$	net upward longwave flux at surface	1	W/m^2
yes	FLNT	$F^S{}_N(P_T)$	net upward longwave flux at top of model	1	W/m^2
yes	FSDS		Flux Shortwave Downwelling at Surface	1	W/m^2
yes	FSNT		net downward solar flux at top of model	1	W/m^2
yes	CLOUD	A_c	cloud fraction	N	fraction
no	SETLWP		prescribed liquid water path	N	g/m^2
no	CLDL		cloud-weighted liquid water path	N	g/m^2

	WP				
yes	EFFCLD	ϵA_c	effective cloud fraction	N	fraction
yes	FLNTC	$F^L{}_N(P_T)_{clr}$	net clearsky upward longwave flux at top	1	W/m^2
yes	FSNTC	$F^S{}_N(P_T)_{clr}$	net clearsky downward solar flux at top	1	W/m^2
yes	FLNSC	$F^L{}_N(P_S)_{clr}$	net clearsky upward longwave flux at surface	1	W/m^2
yes	FSNSC	$F^S{}_N(P_S)_{clr}$	net clearsky downward solar flux at surface	1	W/m^2
yes	OMEGA	ω	vertical pressure velocity	N	Pa/s
no	DQP	$R_{cs}+R_{ls}$	*q* tendency from rainout	N	$(Kg/Kg_{air})/s$
yes	TAUX	τ_λ	zonal surface stress	1	N/m^2
yes	TAUY	τ_μ	meridional surface stress	1	N/m^2
yes	SRFRAD	$F_N^S + F\downarrow($	radiative flux absorbed at the surface	1	W/m^2
yes	QRS	Q_s	solar heating rate	N	K/s
yes	QRL	Q_{lw}	longwave heating rate	N	K/s
yes	CLDTOT	$A^T{}_c$	random overlap total cloud cover	1	fraction
yes	CLDLOW	$A^L{}_c$	random overlap low cloud cover	1	fraction
yes	CLDMED	$A^M{}_c$	random overlap medium cloud cover	1	fraction
yes	CLDHGH	$A^H{}_c$	random overlap high cloud cover	1	fraction
no	TOTL		total liquid water path	1	fraction

	WP					
yes	SOLIN	S_I	solar insolation	1	W/m^2
no	UTEND		*u* tendency	N	m/s^2
no	VTEND		*v* tendency	N	m/s^2
no	TTEND		*T* tendency	N	K/s
no	LPSTEN		surface pressure tendency	1	Pa/s
yes	UTGW	$\left(\frac{\delta u}{\delta t}\right)_{GW}$	gravity wave drag *u* tendency	N	m/s^2
yes	VTGW	$\left(\frac{\delta v}{\delta t}\right)_{GW}$	gravity wave drag *v* tendency	N	m/s^2
yes	TAUGWX	$\left(\tau_{\lambda}\right)_{GW}$	gravity wave drag zonal surface stress	1	N/m^2
yes	TAUGWY	$\left(\tau_{\mu}\right)_{GW}$	gravity wave drag meridional surface stress	1	N/m^2
yes	DTCOND	$\left(\frac{\delta T}{\delta t}\right)_{A}$	*T* tendency from adjustment physics	N	K/s
yes	CMFDT	$\left(\frac{\delta T}{\delta t}\right)_{CC}$	*T* tendency from moist convection	N	K/s
yes	CMFDQ	$\left(\frac{\delta q}{\delta t}\right)_{CC}$	*q* tendency from moist convection	N	$(Kg/Kg_{air})/s$

no	`CMFDQR`		rainout (condensation)	N	$(Kg/Kg_{air})/s$
yes	`CMFMC`	M_c	total convective mass flux	N	$(Kg/Kg_{air})/s$
no	`CMFSL`	$F_{s\text{-}Ll}$	convective liquid water static energy flux	N	W/m^2
no	`CMFLQ`	F_{q+l}	convective total water flux	N	W/m^2
yes	`CNVCLD`	A_{cc}	convective cloud fraction	1	fraction
no	`VT`	vT	Meridional heat transport	N	K * m/s
no	`VZ`	vz	Meridional transport of gravitational potential energy	N	m^3/s^3
no	`VQ`	vq	Meridional water transport	N	m/s
no	`VVPUU`	v^2+u^2	Kinetic energy	N	m^2/s^2
no	`RELHUM`	RH	Relative humidity	N	fraction
no	`Z3`	z	Geopotential height (above sea level)	N	m
no	`SICTHK`		sea-ice thickness; on default file if using the SOM	1	m
no	`MQ`	$q\Delta P$	Water mass	N	Kg/m^2
no	`PSL`	P_{sl}	Sea level pressure	1	Pa

Fields are written to the history buffer with a call to subroutine `outfld` (see Sec. 3.5 and Sec. 4.2 for more details). Due to efficiency considerations, calls to `outfld` for most "inactive" fields (i.e. those not on the default primary history file) are commented out in the code. In order to activate an inactive field either to the primary history file via `PRIMARY`, or to an auxiliary history file via `AUXF`, the user must remove the `C` from column one of the corresponding call to `outfld` for the particular field. The order in which fields are written to the history file is determined at run time via a hashing algorithm. Hence, the order in which the default fields appear in Table 2.13 does not necessarily reflect the actual storage order of fields in the file.

A namelist option is available to produce a monthly-averaged primary history file. By setting `NINAVG(1)='Q'`, the fields on the primary history file (excluding `PHIS` and `ORO` which are always instantaneous) will be averaged over the period beginning from the first timestep of the current month up to and including the last timestep of that month. Each monthly averaged history file will contain exactly one time slice of data, regardless of the values of `MFILT(1)` and `NHTFRQ`. In addition to those fields shown as "on" by default in the Master Field List, a monthly-averaged primary history file includes 14 other fields by default. These fields are shown in Table 2.14. Due to these additional fields, the history buffer will require additional memory whether stored in-core, or out-of-core on the SSD. For a standard T42 resolution, the memory request should be increased by 1Mw for a monthly averaged run over what would be required if the run were not monthly averaged. See "Creating and Running the Executable" for information about specifying memory. As with any field on the history file, the additional fields may be excluded from the primary file using the `EXCLUDE` namelist variable.

Table 2.14: Additional Fields on a Monthly-Averaged Primary History File

Field Name	**Symbol**	**Field Description**	**NL**	**Units**
`VT`	vT	Meridional heat transport	N	K*m/s
`VZ`	vz	Meridional transport of gravitational potential energy	N	m^3/s^3
`VQ`	vq	Meridional water transport	N	m/s
`VVPUU`	v^2+u^2	Kinetic energy	N	m^2/s^2
`RELHUM`	RH	Relative humidity	N	fraction
`Z3`	z	Geopotential height (above sea level)	N	m
`MQ`	$q\Delta P$	Water mass	N	Kg/m^2
`PSL`	P_{sl}	Sea level pressure	1	Pa
`OMEGAUP`	$\omega^{\dagger}$	Average of omega when vertical motion is up	N	Pa/s
`NUMOMGUP`	$N^{\dagger}$	Percentage of time vertical motion is up	N	fraction
`CLOUDUP`	$A_c^{\dagger}$	Average of cloud when vertical motion is up	N	fraction

DPSLON	$\left(\frac{\partial}{\partial \lambda}\ln p_s\right.$	Longitudinal derivative of surface pressure	N	m/s^2
DPSLAT	$\left(\left(1 - \mu^2\right)\right.$	Latitudinal derivative of surface pressure	N	m/s^2

2.5.1.2 Auxiliary History Files

CCM3.6 has the capability to write multiple history files simultaneously. All fields that are declared "active" (default) in the Master Field List appear on the primary history file. The user may declare additional files, known as "auxiliary" history files, via namelist variable `AUXF`. The capability to write auxiliary history files provides the user flexibility in the frequency at which various history data are written. Auxiliary files may contain the same or different fields as compared with the primary history file. These fields may be written on different timesteps (and therefore have different averaging periods), with different packing densities, different averaging type (A, I, M, or X), and with a different number of time samples per file. A user may specify up to four auxiliary history files if `INITHIST` is set to *MONTHLY* or *YEARLY* and up to five auxiliary history files otherwise.

Why would a user want to write an auxiliary history file? Perhaps there are additional fields resulting from a new parameterization which should be reported more frequently than the primary history file fields. The auxiliary file provides a means of doing this while avoiding the waste of resources required to write the primary file more frequently. The use of namelist variable `AUXF` in declaring two auxiliary history files is demonstrated in "Example 5 -- Branch Run" . An auxiliary file may contain any number of fields up to the number of fields on the Master Field List. Fields may be removed from the primary file and placed on an auxiliary file for the added convenience of smaller

2.5.1.3 Naming the Primary and Auxiliary History Files

Primary history volumes which are not monthly averaged will be named with an "`h`" followed by a four-digit number. For example, on a typical initial run, the first primary history file written is named "`h0001`". Once this first disk file has been filled with `MFILT(1)` time samples, it is closed and new primary history file is opened. This new file, the second in the series, is named `h0002`. Subsequent primary history files will be named `h0003`, `h0004`, etc. There are two exceptions to this naming rule. First, if namelist variable `STFNUM` is set, naming will begin with this number. For instance, if `STFNUM=123`, then the first primary history file will be named `h0123`. Second, if the primary history file is monthly averaged, then the filename will be composed of the year and month of the model date. For instance, if the primary history file is being written at the end of the ninth month of year 26, the file name would be `0026-09` (see "Example 8 -- Generate a Monthly-Averaged History File").

Naming of the auxiliary history files is very similar to the primary history file. The first auxiliary file is named with an "ha" followed by a four-digit number. The second auxiliary history file is named with an "hb" followed by a four-digit number, etc. Thus, files named ha0001, ha0002,... will be generated for the first auxiliary history file, and hb0001, hb0002,... for the second auxiliary history file, etc.

2.5.2 Restart Datasets

Sec. 2.4 provides example namelists for initial and continuation runs. There are two types of continuation runs: a restart run and a branch run. A restart run is an exact continuation of a previous run from its point of termination. Namelist variables other than NESTEP, NELAPSE, and NSREST **SHOULD NEVER** be modified for restart runs. A branch run on the other hand is a new case that uses restart data from a previous run to begin the integration. Regardless of the type of continuation run being conducted, information is always obtained from the restart files output by a previous run. A more detailed discussion of the restart files, including a general description of their contents and related coding issues, is presented in "Restart Files" .

Master, primary, secondary, and history buffer restart files comprise the four types of "restart" files. These files contain grid-point data and other information necessary to continue a run. The "restart pointer" file is simply an ASCII text file that holds the full pathname of the most recently written master restart file.

When master and primary restart files are written, secondary and history restart files may also be written. Secondary restart files will be written only if the timestep when restart files are written corresponds to a timestep where absorptivity/emissivity calculations will not be performed. If this is the case, absorptivity/emissivity grid-point values will be saved. History buffer restart files are written only if a history file (either primary or auxiliary) is not written on the same timestep as the primary restart dataset. Restart files are only meant to be read by the model. Only the name of the master restart file is needed as input for a continuation run. The model is able to determine what other restart files (i.e. primary, secondary, etc.) need to be read from data saved on the master restart file.

Primary, secondary and history buffer restart files all have root names of the form ccmr_yyyymmdd_sssss, but with different suffixes appended. In the root name, yyyy corresponds to the year, mm is the month, dd is the day and sssss is the seconds of the date yyyymmdd. The suffix for the primary restart file appears as uppercase letters, such as ".A",".B", etc. Secondary restart files have lower case letters as suffixes, such as ".a", ".b", etc. The history buffer restart files will have a ".h0" appended to the root name, followed by an integer, such as ".1". The first auxiliary history file will have a .ha appended followed by an integer, and so on. The following is an example of the restart files written during a simulation.

- ccmr_100101_3600 - master restart file
- ccmr_100101_3600.A - primary restart file
- ccmr_100101_3600.a - secondary restart file

- `ccmr_100101_3600.ha.1` - history buffer restart file for first auxiliary history file
- `ccmr_100101_3600.hb.1` - history buffer restart file for second auxiliary history file

2.5.3. Mass Store Archiving.

History files and restart datasets archived on the NCAR Mass Storage System are transferred asynchronously to the MSS as they are completed. If namelist variable `IRT` is zero, history files will not be archived. Similarly, if `RIRT` is zero, restart files will not be archived, but retained locally. Mass Store pathnames for these transfers are automatically generated. For history files, the pathname is as follows:

`/USERNAME/csm/CASEID/ccm3/hist/hxxxx`

Here, `USERNAME` is the upper-case equivalent of the user's login name, i.e., the user's root directory on the Mass Store System, and `CASEID` is the case identifier and is set via the namelist input. For restart datasets, the following pathname is used:

`/USERNAME/csm/CASEID/ccm3/rest/ccmr_yyyymm_sssss`

It is recommended that the user specify a non-blank write password using the namelist variable `NSWRPS` (see Table 2.5). File passwords are the only form of security available on the Mass Storage System. If the write password is not set, any other user can overwrite or change the files after they have been archived..

The size of history files may be reduced by using the namelist packing feature (see `NDENS` in Table 2.5). This feature is only available on the NCAR Cray machines.

2.6 Getting Help

If a user should encounter bugs in the code (i.e., it doesn't behave in a way in which the documentation says it should), the problem should be reported electronically to **ccm-bugs@ucar.edu**. When reporting a suspected bug, please include the following information: 1) the architecture on which the code was built; 2) horizontal resolution; 3) ocean configuration 4) namelist; 5) run script; and 6) Model printout.

2.6.1 The CCM Web Page

On-line copies of this Users Guide and other CCM documentation may be found on the CCM home page at **http://www.cgd.ucar.edu/cms/ccm3**. The CCM3 web site will also contain information on software updates to the CCM, bug fixes, notices of meetings or upcoming events, and so forth.

2.6.2 The ccm-users e-mail Group

The ccm-users discussion group is an open e-mail forum for rapid exchange of information, ideas, and topics of interest relating to the various versions of the NCAR CCM (Community Climate Model). This includes sharing software tools, datasets, programming tips and examples, as well as discussions of questions, problems and workarounds. The primary motivation for the establishment of this list is to facilitate and encourage communication between the users of the CCM around the world. This mail group will also be used to distribute announcements related to the CCM and the CCM Processor.

Documented bug reports should be reported to ccm-bugs@ncar.ucar.edu, not the ccm-users mail group.

To SUBSCRIBE, send mail to MajorDomo@ucar.edu which contains the following in the body of the message:

subscribe ccm-users

You will receive confirmation via e-mail and you will be automatically subscribed to the list.

To UNSUBSCRIBE, send mail to MajorDomo@ucar.edu which contains the following in the body of the message:

unsubscribe ccm-users

To POST: Once subscribed, send messages to: ccm-users@ucar.edu

To CHANGE YOUR EMAIL ADDRESS: Simply unsubscribe from your old address and re-subscribe from your new address. This prevents error messages and the necessity to manually verify your address.

3. CCM3.6 Internals

In this section, code is described for those users who seek to modify it or add their own physical parameterizations.

3.1 Design Philosophy

Large numerical model codes are notorious for being difficult to read, understand, and modify. In 1989, an international committee of geophysical modelers wrote some coding rules designed to address these problems, with a specific emphasis on making parameterizations "plug-compatible". Their paper, "Rules for Interchange of Physical Parameterizations" (Kalnay *et al.*, 1989), provided valuable guidance in the design of the previous Community Climate Model, CCM2 (Bath et. al., 1992). Use of these coding rules was continued in CCM3 and CCM3.6.

The control structure of CCM3 with respect to initialization, time integration, and gridpoint to spectral transformations is similar to that of CCM2 Some changes were made in CCM3 to allow a straightforward coupling of the atmospheric model to land, ocean, and sea ice components. For example, the latitude loop which invokes the physical parameterizations was split into two routines to facilitate exchange of flux information between models. Modifications were also made in CCM3 and CCM3.6 to allow the code to be run on architectures other than Cray PVP (parallel vector processor) machines. In addition to Cray PVP machines, CCM3.6 is supported on SGI, SUN, and HP architectures in both shared-memory and distributed-memory (MPI) variants. Hooks also exist for T3D/E and RS6K machines, but the former no longer works with the migration to netCDF (described below), and the latter is no longer supported because RS6K platforms are no longer available to the NCAR Climate Modeling Section.

One important change from CCM3 to CCM3.6 is that the binary format of most input datasets was changed to the self-describing binary format netCDF. One major advantage of netCDF is that the binary format is independent of the machine on which the code is being run.

CCM3 has retained the potential to be run with some data structures stored out-of-core. This feature is probably only of value on machines which provide an extremely fast I/O device, such as the Cray solid state storage device (SSD). Thus, the model also has the capability of retaining some or all of the main model buffers in-core. Namelist variables which control the in-core or out-of-core nature of these buffers are `INCORHST`, `INCORBUF`, and `INCORRAD`. As in CCM2, the main model buffer of CCM3 consists of variables that must be carried at more than one time level, more than one latitude scan, or for storage contiguity purposes.

In CCM3, memory is managed with the following goals in mind:

- Use memory efficiently, mainly to run economically at higher resolutions.

- Ensure that data access is convenient and understandable at the physical parameterization level.
- Accommodate parallel processing at the latitude loop level.
- Take advantage of contiguous storage while performing I/O during the out-of-core implementation.

To accommodate the semi-Lagrangian transport code, certain variables need to be retained in main memory as three-dimensional arrays (dimensioned longitude by vertical level by latitude) at two time levels. These arrays reside in named common `/com3d/`, and therefore reside in main memory throughout a run.

Temporary storage, such as that for Fourier coefficients during the spectral transform, is maintained as locally declared arrays on the "stack" (the automatic dynamic memory management mechanism). These data structures are described in "Data Structures" .

Within the time integration loop inside `stepon`, only the control routines `linemsbc`, `linemsac`, and `spegrd` interface directly with the model buffers. At the beginning of these routines, Cray pointers are used to associate fields in the model buffers with mnemonically named field arrays. The physical parameterization routines called by these control routines need only pass these fields through argument lists, without clumsy indexing directly into the buffers. Plug-compatibility is attained through mnemonics and by using the argument list instead of common blocks wherever possible for the communication of data between routines.

Most of the parallel processing is accomplished at the latitude loop level, with an arbitrary number of processors sharing the work. Data structures not dimensioned by latitude are allocated on the stack, so that each processor has its own copy of writable memory for these arrays. I/O to the out-of-core scratch files is implemented as Fortran direct access, synchronous I/O to accommodate the random order of latitude loop execution. If running multitasked, all data accumulations, e.g. Gaussian quadrature, occur in the same order as if running single-threaded to guarantee identically reproducible results for multiple runs.

The CCM3 history file interface is designed so users can easily record whatever data they like on the output history files. Fields that appear in the Master Field List, generated at initialization time by subroutine `bldfld`, may be included on the history file either by default or by namelist request. A field is recorded in the history file buffer by calling subroutine `outfld`.

Provision is made for recording instantaneous or averaged, as well as maxima or minima. Fields are written to the history file within a multitasked loop over latitudes. Due to multitasking and the resulting random order of execution of the latitude loop, the sequentially written history file may be randomly ordered by latitude. Thus, each latitude record contains the latitude index as its first word. Indices increase from south to north.

3.2 Model Code Flow

This section provides a narrative of the code flow through the initialization phase and the driving computational loops. Refer to the calling tree in "Appendix B: CCM3.6 Calling Tree". The user may also wish to review "Overview of CCM3.6" before reading this section. Details of the physical parameterizations are not presented here -- see Kiehl e*t al.* (1996).

3.2.1 Initialization

The first tasks performed are defining the logical units used for I/O (subroutine `lunits`), presetting namelist variables, (subroutines `preset, lsmctli`) reading in the Fortran namelist data (`subroutines data, lsmctli`), and setting pointers and lengths associated with the main model buffer (`subroutine points`).

From this point on, initialization takes one of two paths. Initial runs are set up in subroutine `inital`, or continuation runs are initialized in subroutine `resume`. These two paths are roughly parallel in nature and call many of the same routines. The primary difference is that an initial run reads the initial dataset and spectrally truncates prognostic fields (see `inidat`, below). A continuation run obtains these fields from the appropriate restart files.

The primary job of subroutine `inidat` is to read the required fields from the initial dataset and spectrally transform the surface pressure (`PS`), wind (`U,V`), temperature (`T`), and surface geopotential fields (`PHIS`). Other required fields are the surface type flag (`ORO`), the sea ice layer temperatures (`TS1` through `TS4`), and the moisture field (`Q3`). If the slab ocean model is enabled, snow depth (`SNOWH`) and sea ice thickness (`SICTHK`) are also read in from the initial dataset.

The amount of local data (stack-based) used in `inidat` is quite large. Three-dimensional arrays exist for vorticity, divergence, and the work array required by the **FFT (Fast Fourier Transform)** package. The need for three dimensional arrays is partly due to the fact that the model can start from an initial dataset randomly ordered in latitude. Since local memory is allocated on the stack (see "Shared-Memory Management"), the large size of the local database does not increase total memory utilization since the local workspace in `inidat` will disappear before the time integration begins.

3.2.2 Time Integration

Subroutine `stepon` controls time integration. After the initialization phase is complete for either an initial run or a continuation run, `stepon` loops through time until completion of the simulation, incrementing variable `nstep` after each timestep. If the run terminates normally, subroutine `scan1bc` will be the last routine called by `stepon,` as well as the first to be called by `stepon` upon restart. The reason `scan1bc` is called at the end of a run is to write the fully time-filtered data to the history file, completing a time accumulation period.

Subroutine `advnce`, called from `scan1bc` each timestep, updates current time information (`calendr`) and time-variant boundary dataset information (subroutines `sstint` or `somint,` and `oznint`).

Current simulation time information is computed in `calendr`. **The definition of a "year" is**

365 days. There are no leap years.

3.2.2.1 Latitude Scans

There are two multitasked, Gaussian latitude scan loops: one in routine `scan1bc` and another in `scan1ac`. These were in one routine, `scan1`, in earlier versions of the Community Climate Model. The routine was split into two pieces in CCM3 in order to accommodate coupling to other geophysical models, such as ocean, and sea ice models through the Climate System Model (CSM) flux coupler (Byran et. al. 1996). The `bc` in `scan1bc` refers to "before coupling", and the `ac` in `scan1ac` to "after coupling". Adjustment physics, cloud calculations and radiative transfer routines are called on the `bc` side, and other physical parameterizations are called on the `ac` side. If coupled to other models, surface fluxes are exchanged between calls to `scan1bc` and `scan1ac`.

Subroutine `lsmdrv` drives the Land Surface Model calculations (Bonan, 1996). These calculations are not multitasked by latitude band, but rather as one big vector which is divided into several subsections of nearly equal length. Computations associated with each of these subsections are then distributed among the available processors. Refer to the document LSM User's Guide for more details.

Subroutine `somoce` drives the slab ocean model calculations, if the SOM option is enabled. The **cpp** token `COUP_SOM` must be set with a `#define` in `misc.h`.

Subroutine `scanslt` drives the semi-Lagrangian transport (SLT) calculations. This is the only latitude scan in which the full dimensionality of the "extended grid" (`plond` and `platd`) is required. In addition to the physical data dimensions (`plon` and `plat`), the extended grid includes additional array storage before the start and after the end of the physical data in both the longitudinal and latitudinal dimensions. The extended grid must be initialized at each timestep. This is done in `sltini`. In `scanslt`, the wind field at time level "`n`" is used to predict the evolution of moisture and constituent fields, from time level `n-1` to time level `n+1` using a semi-Lagrangian transport scheme *(*Williamson and Rasch, 1989). No I/O is done in `scanslt`, since the prognostic arrays `u3`, `v3`, `wfld`, and `q3` are entirely in-core. The local database of subroutine `scanslt` contains numerous multi-dimensional arrays. Since many of these arrays contain a constituent index, memory use can increase dramatically as the number of transported constituents is increased.

Routine `sltini` fills the extensions of certain arrays used by the SLT package in the longitudinal and latitudinal dimensions. Only a small amount of computational work is performed in `sltini`, but the routine is multitasked over latitude bands since it is called every timestep.

Subroutine `scan2` primarily converts spectral space prognostic variables back to gridpoint space and computes global integrals of dry and moist mass. The routine multitasks over latitude **pairs** rather than individual latitudes. The last step in `scan2` toggles the time indices `n3` and `n3m1` after the conversion back to gridpoint space. No copying of the actual data is necessary. Prognostic data that were referenced by the current time index (`n3`) will become the previous time index (`n3m1`) while data just computed in time index `n3m1` are referenced by the "next" (`n3`) time index at the start of the next iteration in `stepon`.

3.2.2.2 Spectral Space Computations

Gaussian quadrature (`subroutine quad`), completion of the semi-implicit timestep (subroutine `tstep`), and horizontal diffusion calculations (subroutine `hordif`) are all accomplished within multitasked loops in `dyndrv`. The quadrature and semi-implicit timestep computations are parallelized over total wavenumber "`n`" when the cpp token PVP is defined, and over Fourier wavenumber "`m`" otherwise. Horizontal diffusion calculations are parallelized over the vertical index "`k`".

3.2.2.3 Computational Driving Routines for Each Latitude Band

Routines `linemsbc` and `linemsac` sit beneath the multitasked latitude loops in `scan1bc` and `scan1ac`, respectively. These routines allocate memory for the main model buffers and history buffer, apply the SLT "fixer", time filter, and call the adjustment physics and tendency physics driving routines.

Routine `aphys` applies adjustments directly to the prognostic variables rather than computing tendencies. These adjustments include dry adiabatic adjustment, moist convection, and large-scale condensation.

Routines `tphysbc` and `tphysac` (called from `linemsbc` and `linemsac`, respectively) call the various physical parameterizations which are applied as tendencies. These include cloud computations, radiation, surface temperature and flux calculations, the planetary boundary layer scheme, vertical diffusion, and gravity wave drag. Cloud calculations are needed internally only by the radiation code. Since radiative computations are normally not performed on every timestep, cloud calculations are performed only on timesteps that the radiative transfer code is exercised.

Routine `radctl` controls both the longwave and shortwave radiative transfer code. While the rest of the code uses *mks* units, *cgs* units are used throughout the radiation computations. Timesteps during which the longwave absorptivity and emissivity calculations are performed is controlled by namelist variable `IRADAE`. Since absorptivities and emissivities can only be calculated on a longwave radiation timestep, `IRADAE` must be an even multiple of `IRADLW`. If `IRADAE` is greater than `IRADLW`, absorptivity and emissivity (a/e) values are stored in a buffer and read into memory during non-absorptivity radiation timesteps. It is recommended that the history file write frequency, `NHTFRQ(1)`be an even multiple of `IRADAE` to ensure that the absorptivities and emissivities need not be written to the restart dataset.

Routine `sltb1` drives the SLT constituent forecast calculations. Each call to this routine fills one latitude line of three-dimensional array `qfcst`. For more detail on the algorithms of the SLT, refer to (Kiehl *et al.*, 1996) .

3.3 Data Structures

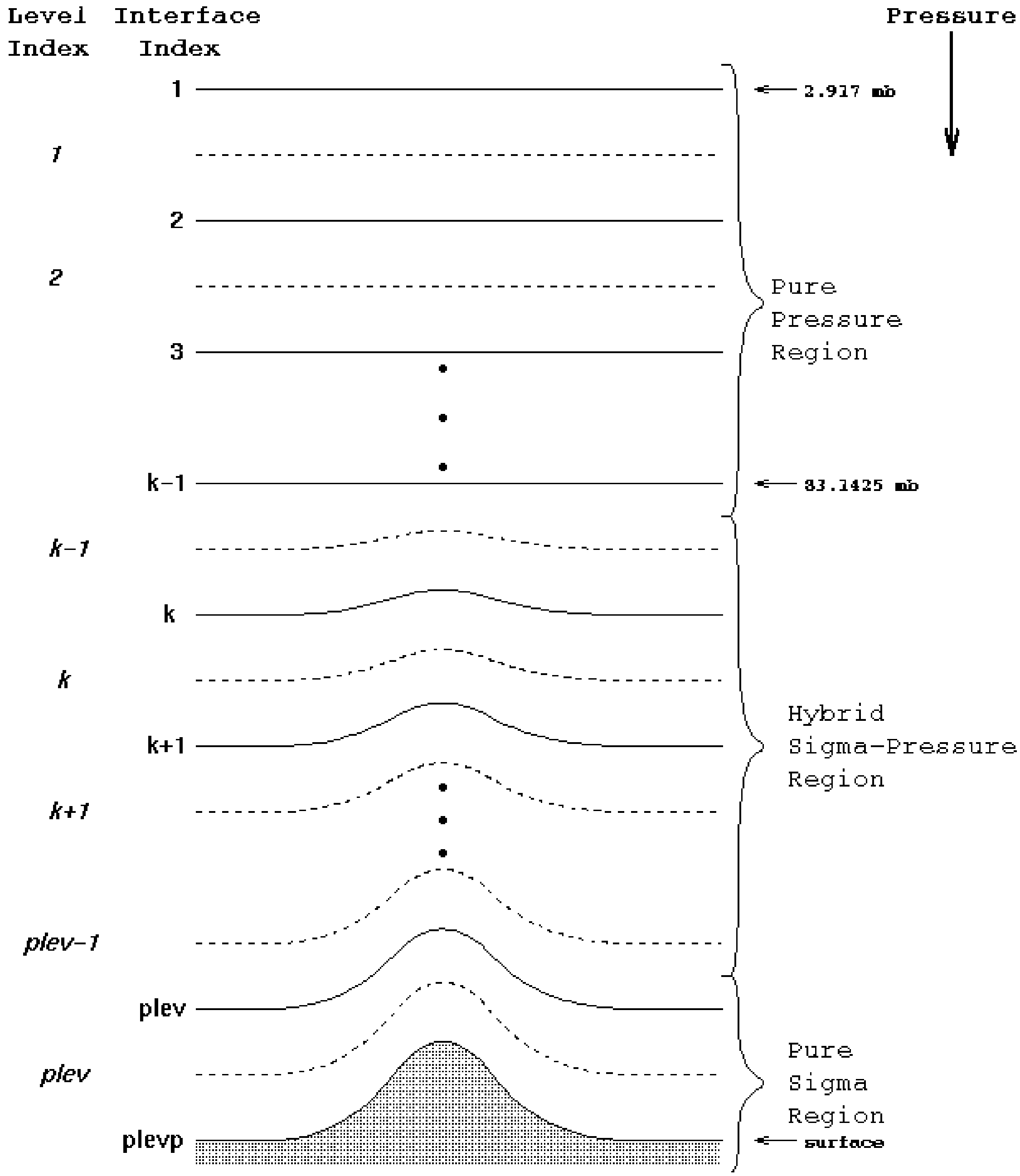
This section discusses design philosophy and implementation techniques for both in-core and out-of-core gridpoint and spectral space data structures. Some special array indexing constructs are also addressed.

CCM3.6 in its default configuration is fully in-core. It is possible to run the model in a partially out-of-core mode by setting namelist variables `INCORBUF` , `INCORRAD` , and/or `INCORHST` to false. In this case certain data structures are resident on a file system, with individual latitude bands of data shuffled into and out of memory as they are needed. This option is only of practical value when an extremely fast I/O device is available, such as the solid state disk (SSD) from Cray Research, Inc. The data structures which have the flexibility to be maintained either in-core or out-of-core based on namelist input are the main model buffers, the radiation buffer, and the history buffer. For the in-core implementation, I/O calls to the SSD are replaced by the setting of Cray pointers to the address of the relevant latitude band of each in-core data structure.

3.3.1 Vertical Coordinate

The vertical coordinate is a hybrid sigma-pressure system. In this system, the upper regions of the atmosphere are discretized by pressure only. Lower vertical levels use the sigma (i.e. $\mathbf{p/p_s}$) vertical coordinate smoothly merged in, with the lowest levels being pure sigma. A schematic representation of the hybrid vertical coordinate and vertical indexing is presented below.

Figure 3.1 Hybrid Vertical Coordinate.



Pressure is defined as:

$$\mathbf{p_{(i,j,k)} = A_k P_0 + B_k P_s(i,j)}$$

where **p** is the pressure at a given level and latitude, longitude grid point. The coefficients **A**, **B** and $\mathbf{P_0}$ are constants. $\mathbf{P_s}$ is the model's current surface pressure. $\mathbf{P_0}$ is set in the model code. The input model initial conditions dataset sets **A** and **B** through the variables **hyam, hyai, hybm**, and **hybi.** The subscript "i" refers to interface levels, and "m" refers to the mid-point levels. "**hyam**" then refers to Hybrid level "A" coefficient on the interfaces. The above formula is also reported on the initial conditions file.

More details on the theoretical nature of the vertical coordinate system can be found in Hack *et al.* (1993).

3.3.2 Gridpoint Data Structures

Prognostic variables held in gridpoint space (as opposed to spectral space) and many arrays associated with the semi-Lagrangian transport scheme (SLT) are maintained in-core. Other gridpoint fields exist either in one of the three buffers (main model buffers, radiation buffer, history buffer), or as local workspace in individual routines. All gridpoint data are stored with longitude as the fastest varying (innermost) subscript, followed by (if applicable) level, constituent, latitude, and finally time. **Longitude indices start at Greenwich and proceed from west to east around the globe. Level indices run from the top of the atmosphere to the bottom. Latitudinal ordering is from south to north.**

The term "constituent" used above refers to water vapor plus an arbitrary number of user-defined advected or non-advected species. The number of advected constituents, defined by Fortran parameter `pcnst`, will always be at least one since water vapor is always transported in the default configuration. The number of non-advected constituents is defined by Fortran parameter `pnats.` The variable `lat` is used throughout to represent the current latitude index. Two time-level indices are necessary for the prognostic variables due to the leapfrog time integration scheme.

Gridpoint arrays are almost universally declared with longitude dimension `plond`, where `plond = plon + 1 + 2*nxpt`. Parameters `plond`, `plon`, and `nxpt` are all defined in header file `pmgrid.h`.

Parameter `plon` is the number of actual data points in the longitudinal direction. `nxpt` is a wrapping number required by the SLT package. Arrays that go through the Fourier transform require two additional longitude points to accommodate storage of the wave 0 (mean) information. Arrays used by the SLT code require at least 1 additional longitude point at the beginning of most arrays, and at least 2 additional points at the end. To avoid proliferation of a dizzying assortment of longitude dimensions, `plond` is used wherever possible, whether or not the full dimensionality is required.

Actual data start at longitude location `1 + nxpt` for all of the prognostics except surface pressure (zonal and meridional wind components, temperature, moisture) and location 1 for all other non SLT-specific arrays. Surface pressure data (in `/com3d/`) start at location 1 because this array is not used in the SLT package. This difference in starting location of the physical data poses an indexing problem for non-SLT routines which operate in gridpoint space. It is undesirable to have to keep track of where the physical data start in every array. The solution chosen was to pass the address of where the data start to non-SLT routines at a very high level in the calling tree. For example, consider the call to `linemsbc` from `scan1bc` (this is very high in the calling tree: refer to Figure B). Some of the calling and declaration sequence looks like:

scan1bc:

```
   call linemsbc(...,ps(1,lat,n3m1),
$                u3(i1,1,j,n3),...)
```

linemsbc:

```
    subroutine linemsbc(...psm1,u3,...)
    real psm1(plond)
    real u3(plond,plev)
```

In `scan1bc`, each argument passed to `linemsbc` represents the starting location of the physical data for that field (i.e. the variable `i1` is equal to `1 + nxpt`, which is the starting index for the physical data in array `u3`). Thus all references to fields in `linemsbc` and below can begin at array element 1. The user need not be concerned with offsets unless they are working with code at the `scan1bc` level or higher where the full data structures are available.

Analogous to the longitudinal case, additional nonphysical gridpoints are needed by the SLT package in the latitudinal dimension. Some arrays are therefore dimensioned `platd`, where `platd = plat + 2*nxpt + 2*jintmx`. Fortran parameter `plat` is the number of actual latitudes, while `nxpt` and `jintmx` are extensions beyond the southernmost and northernmost physical locations (as defined in `pmgrid.h`). The starting latitude index for physical data in these arrays is `1 + nxpt + jintmx`. Routine `scan1bc`, for example, passes the array addresses corresponding to the latitude indices for which physical data actually exist.

In the above example, at the level of `linemsbc` latitude indices are no longer present because all computations in `linemsbc` and below are independent of latitude. A similar approach is used with the array that contains water vapor plus other advected constituents. In the radiation code for example, water vapor is the only relevant constituent. Therefore, array `h2ommr` in the radiation package has no constituent index.

3.3.2.1 Model Buffers

The main model buffers contain longitude-level slices of a number of fields stored contiguously. In an out-of-core configuration, these arrays are cycled through an I/O device, latitude band by latitude band. A schematic representation of this buffer is presented below.

Figure 3.2.Layout of main model buffers, `b1` and `b2`.

Listed within each buffer is the variable holding the storage location of its associated field. To the right of each variable is a description of the field including whether the field is single-level(s) or multi-level (m). Also shown are related parameters pflenb, pflena, pbflnb, pbflna, and plenalcl.

Buffer b1(pflenb)

	variable	
pbflna	Nzt1	vorticity: half time-filtered (m)
	Ndt1	divergence: half time-filtered (m)
	Nphis	surface geopotential (s)
	Nqminus (PCNST)	q being advected in slt code (m)
	Npsm2	surface pressure, time n-2 (s)
	Num2	u-wind, time n-2 (m)
	Nvm2	v-wind, time n-2 (m)
	ntm2	temperature, time n-2 (m)
	nqm2 (PCNST)	constituents, H2O first (m)
	npb1ht	PBL height (s)
	nqpert (PCNST)	PBL moisture/constit. Perturb. (s)
	ntpert	PBL temperature perturb. (s)
	nfsns	surface absorbed solar flux (s)
	nqrs	shortwave rad. heating rate (m)

plenalcl

Buffer b2(pflena)

	variable	
pbflna	Npsp1	sfc. pressure, time n(s)
pbflna	Nzp1	vorticity, time n (m)
pbflna	Ndp1	divergence, time n (m)
pbflna	Ndpthp1	horiz. temp. diffusion (m)
pbflna	Ndpslp1	long. deriv. of lpns, time n (s)
pbflna	Ndpsmp1	lat. deriv. of lpns, time n (m)
	Nduhp1	horiz. u-momentum diff . (m)
	Hdvhp1	horiz. v-momentum diff. (m)
	ndpsp1	pressure gradient (s)

	nqrl	Longwave rad. heating rate (m)	
p	npsm1	surface pressure, time n-1 (s)	
b	nzm1	vorticity, time n-1 (m)	
f	ndm1	divergence, time n-1 (m)	
l	not used	(m)	
n	not used	(s)	
a	not used	(s)	

These buffers hold only those fields which must be carried across discrete time levels, that are needed in more than one Gaussian latitude scan, or that need to be contiguous to other fields. One example of a field which must be carried from one time level to the next is the shortwave radiative heating rate (`qrs`), which must be passed forward in time between calls to the radiation routines.

Surface pressure, vorticity, and divergence are in the buffer for contiguity purposes. Contiguity is required for optimal vectorization of the **FFT** package. This is important since the **FFT** routines used in CCM3.6 vectorize over the number of transforms being done. By putting surface pressure next to vorticity and divergence, an 18-level model produces a respectable vector length of 37 (i.e. `2*18 + 1`) in the Fourier transform of these fields. If surface pressure were transformed separately, the vector length for this field would be 1, which would significantly degrade performance. Surface pressure is contained in memory (common `/com3d/`), and is copied to and from the model buffer for the sole purpose of avoiding the **FFT** performance penalty.

All buffer fields are accessed computationally as individual, appropriately dimensioned arrays. This approach yields readable code, and actually runs a bit faster than using direct buffer references throughout. It also enables usage of the array bounds checker (`f90 -Rb` on Cray machines). This functionality is obtained through Cray Fortran pointer variables, as illustrated in a section of subroutine `spegrd` below:

```
real b2(plenalcl)    ! model buffer
pointer (pz,z)       ! declaration of pointer variable
real z(plond,plev)   ! part of model buffer
   .
   .
   .
pz = loc(b2(nzp1+1)) ! array z may now be accessed
```

Array `z,` which is a part of a larger buffer, can be referenced as a separate two-dimensional array once the Cray pointer is set. Though not part of the ANSI Fortran standard, Cray pointers are supported on a wide variety of platforms.

3.3.2.2 History Buffer

Data written to the history files are maintained in a separate buffer distinct from the main model buffers. This buffer may be cycled to the SSD (unit `sunit`) or maintained in-core. In either case its existence must span several time levels because most fields written to the history file are time-averaged. Therefore, the data must be carried across time levels between the `outfld` calls which accumulate them into the history buffer. The history buffer is maintained as a singly-dimensioned array, with integer pointers delimiting locations of the various fields within the history buffer array. Refer to "Details of CCM3 Input and Output Datasets" under "History Buffer" for more information.

3.3.2.3 Absorptivity/Emissivity Arrays

Longwave radiation calculations of absorptivities and emissivities for various gases are referenced latitude band by latitude band, and can be held in-core at all times or cycled to an out-of-core work unit. In the default configuration, the absorptivities and emissivities are computed and written out every 12 simulated hours and read in as input every hour. Nonadjacent layer absorptivities, nearest-layer absorptivities, and total emissivity arrays are stacked in array `absems` in order to minimize the number of I/O calls in the out-of-core configuration.

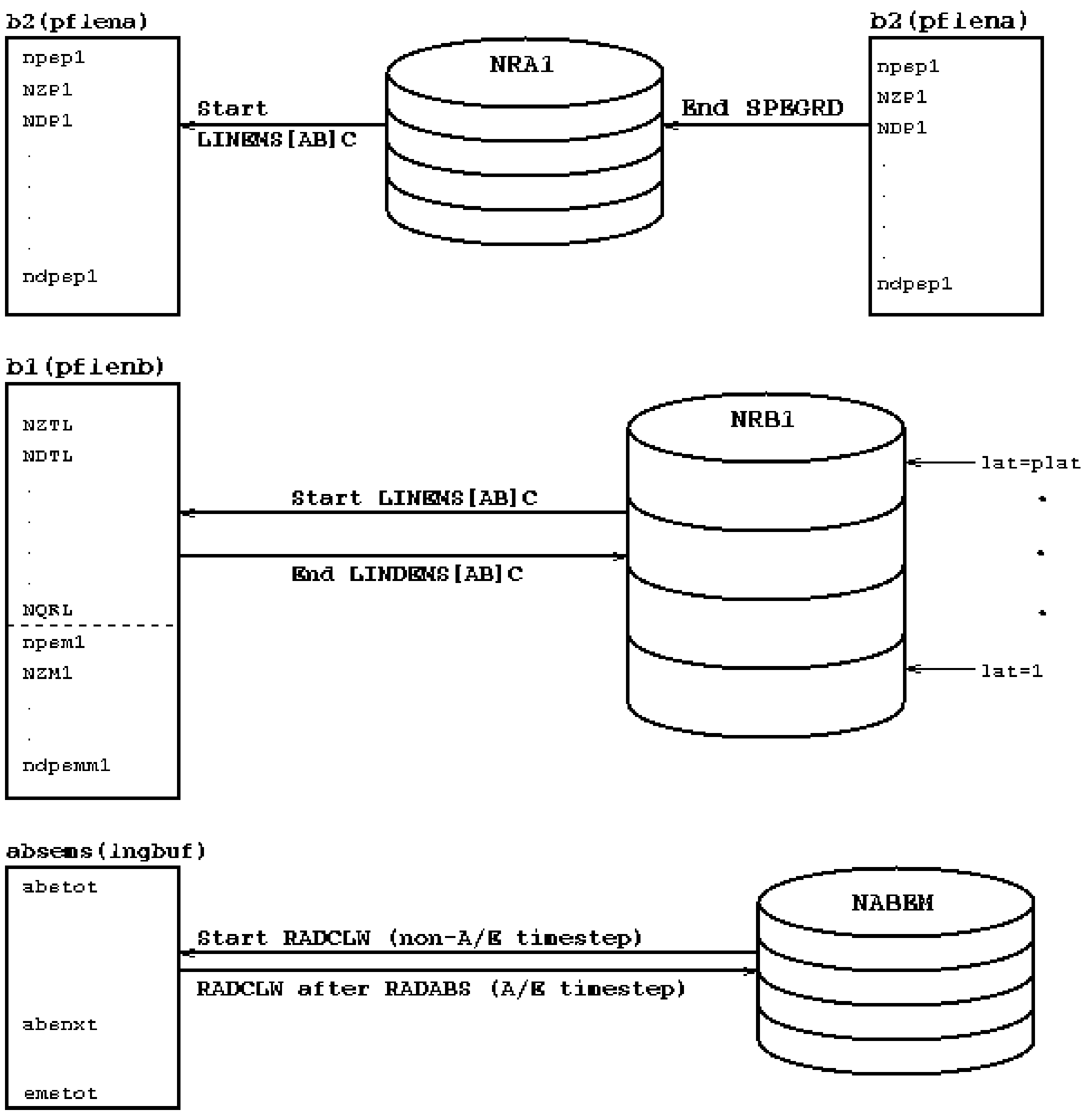
It is recommend that the user choose a primary history file write frequency (namelist variable `NHTFRQ(1)`) that is an even multiple of the longwave radiation calculation frequency(`IRADAE`). Otherwise, the large absorptivity/emissivity arrays must be written to the restart dataset.

3.3.2.4 Out-of-Core Data Storage: The SSD Work Units

Data flow to and from the out-of-core units is depicted in Figure 3.3 for the main model buffers and the radiation buffer. The buffers are portrayed by vertical rectangles, and the out-of-core work units by the multi-layered cylinders. Each layer of a cylinder represents a latitude slice of data. The arrows connecting the buffers to the out-of-core units show the specific multitasked routines that perform the I/O. `nra1`, `nrb1`, and `nabem` are the Fortran unit numbers associated with the out-of-core work units. On startup, subroutine `inidat` does the initial copy of data from the main model buffers to units `nra1` and `nrb1`. Data flow between the history buffer and out-of-core work unit `sunit` is analogous to the main model buffers. All I/O is synchronous. In addition to being simpler and easier to understand than an asynchronous double-buffering scheme, single-buffered, synchronous I/O allows the use of standard Fortran **read** and **write**, which makes porting the code to different computer architectures an easier task. On Cray machines, CCM3.6 uses the UNICOS "Secondary Data Segment" (SDS) file interface on the SSD for the work files required by the out-of-core implementation. These files are written as Fortran direct access files, where each record represents a latitude band of data. All I/O to the out-of-core work units is done as unblocked files. Record size is computed by rounding up to the nearest multiple of 512 words. This saves an enormous amount of system CPU time which would otherwise have to be spent blocking and unblocking the data records for the user.

Figure 3.3.Use of SSD Work Units.

Arrows indicate movement between model buffers (vertical rectangles) and the out-of-core word units (layered cylinders). The variable name holding the Fortran unit number of each out-of-core unit is shown on the top layer of each cylinder. Each arrow is labeled with the subroutine name responsible for the data transfer.



3.3.2.5 Local Workspace

Nearly all local workspace declared in individual routines is stack-based such that variables declared local to a given subroutine exist only during the lifetime of that routine, disappearing upon execution of the **return** statement. This approach has the advantage of allowing the user to declare workspace wherever it is required, without having to worry about clobbering space that another routine needs. A stack-based memory management scheme is vital for multitasking considerations. This topic is discussed in more detail in "Multitasking Strategy" .

Space for locally declared variables is allocated statically (i.e. not on the stack) only for those variables which are initialized with a **data** statement, or explicitly mentioned in a **save** statement. When a variable is allocated statically, its value is preserved between calls to the routine in which it is declared.

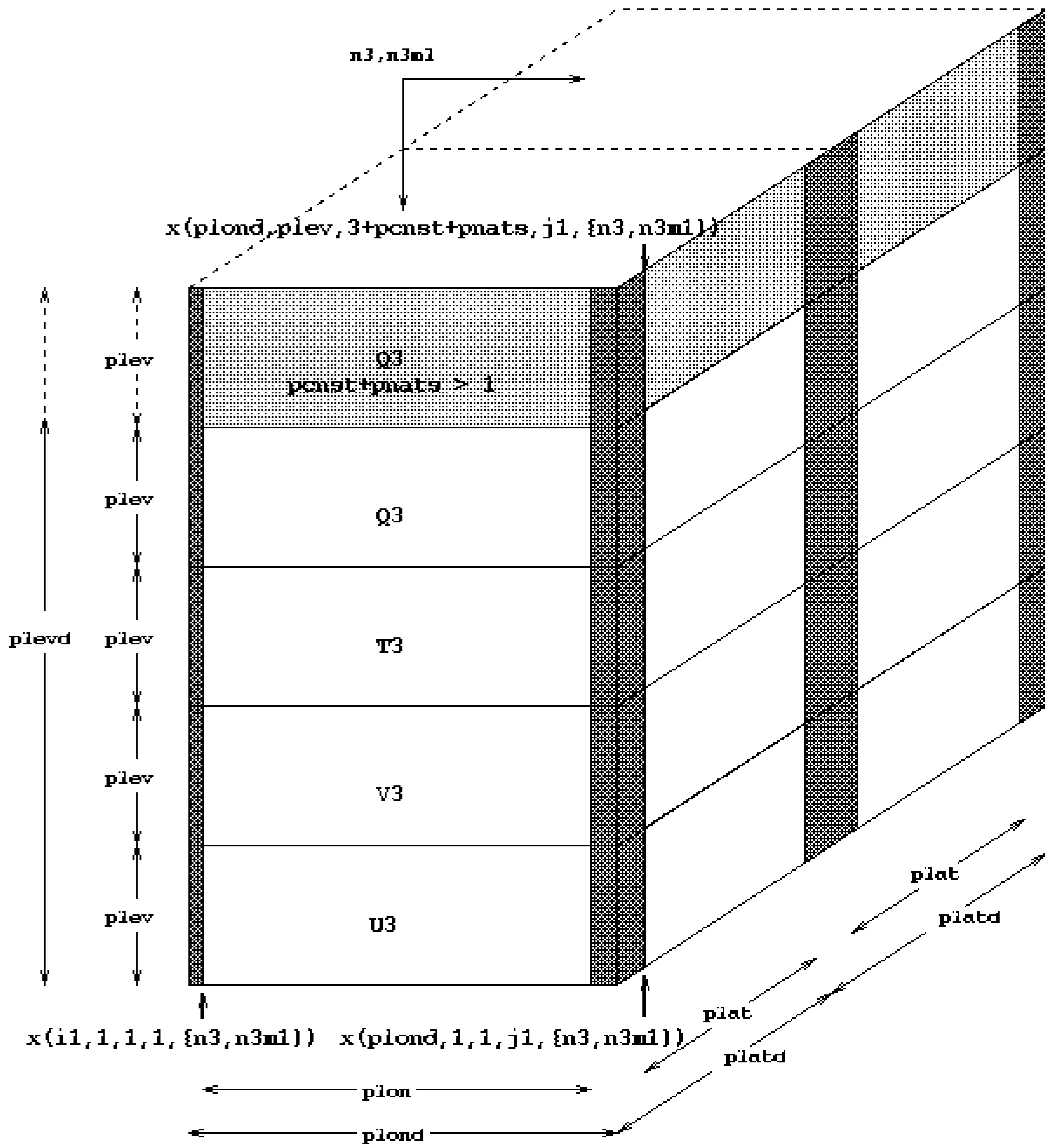
3.3.2.6 In-Core Grid-point Arrays

Prognostic variables surface pressure, zonal wind, meridional wind, temperature, and water vapor (hereafter referred to by their model variable names `ps`, `u3`, `v3`, `t3`, and `q3` respectively) are kept in memory at all times (common `/com3d/`). Historically this was done primarily for the benefit of the SLT code, since this algorithm requires data simultaneously at a number of longitudes and latitudes. More recently, the use of global data structures has proven advantageous given the advent of machines with very large memories.

A schematic view of the data structure defining `u3`, `v3`, `t3`, and `q3` is shown in Figure 3.4. This storage arrangement accomplishes two important goals. The contiguity of fields `u3`, `v3`, and `t3` results in a more efficient Fourier transform for the same reasons mentioned earlier for surface pressure, vorticity, and divergence in the main model buffer. Additionally, the SLT algorithm takes advantage of the contiguity of `u3` and `v3` to produce more efficient code.

Figure 3.4 The /com3d/ storage arrangement.

Dark shaded regions represent non-physical data points. Light shaded region represents the potential for additional transported constituents. All names are as defined in the CCM3.6 code.



If `pcnst+pnats` is greater than 1 (that is, at least one user-defined constituent is declared), the longitude-level cross sections of the additional constituents are stored adjacent to water vapor. Access to the constituent array at a high level in the calling tree is of the form `q3(i1+i-1,k,m,j1+lat-1,n3)`, where `m` is the constituent index, `lat` is the south-to-north latitude index, and `n3` is the time level index. `m=1` refers to the moisture field, with additional constituents stored in locations `m=2,pcnst+pnats.` Variables `i1` and `j1` are the longitudinal and latitudinal offsets of the first "real" data elements, with the data values outside the domain used by the SLT code and the FFT package. Much of the complexity of this data structure can be hidden by passing array offsets to lower level routines.

Since u3, v3, and t3 are a part of this same data structure, a dimensioning construct was employed to avoid having to include a constituent index on these fields. Defining the Fortran parameter plevd, as plev*p3d (where p3d=3+pcnst+pnats), allows u3, v3, and t3 to be dimensioned as (plond,plevd,platd,2). To see how this is used we examine the (/com3d/) common block :

```
common/com3d/n3m1,n3
   common/com3d/ps(plond,plat,2),x(plond,plev,p3d,platd,2)
C
   real u3(plond,plevd,platd,2),       ! u-wind component
  $     v3(plond,plevd,platd,2),       ! v-wind component
  $     t3(plond,plevd,platd,2),       ! temperature
  $     q3(plond,plev,p3d,platd,2) ! specific humidity
                                       ! and constituents

   equivalence (u3,x(1,1,1,1,1)),
  $            (v3,x(1,1,2,1,1)),
  $            (t3,x(1,1,3,1,1)),
  $            (q3,x(1,1,4,1,1))
C
   integer n3m1,n3 ! time index pointers
   real ps, ! surface pressure
  $     x   ! contiguous u,v,t,q
```

The 3 in the 3+pcnst+pnats dimension accounts for prognostic variables u3,v3, and t3. "n3" and "n3m1" are time level indices used as the last subscript in the prognostic arrays. Each holds the value 1 or 2, with the values toggled after each timestep to avoid a memory transfer of time n-1 data. Equivalenced array "x" allows reference to all four three-dimensional prognostic arrays as a single entity.

In addition to the prognostics, SLT three-dimensional fields lammp, phimp, sigmp, and qfcst are held in-core and given global scope by keeping them in a common block (/comslt/). These arrays are held in common rather than kept local to scanslt because they need to be written to the restart dataset and they are required in other scan loops.

Other three-dimensional arrays held in memory, but which are not written to the restart dataset are local scanslt arrays uxl , uxr , qxl , and qxr . The SLT scheme may transport an arbitrary number of constituents. It is important to realize that in-core arrays q3 , qfcst , qxl , and qxr have a constituent dimension, so transporting a large number of constituents can lead to rapid growth in memory use.

3.3.3 Spectral Data Structures

Spectral coefficient arrays and Legendre polynomials are kept in memory in common block /comspe/. The spectral coefficients for vorticity, divergence, temperature, moisture, and log surface pressure are maintained as individual one-dimensional arrays (vz, d, t, q, and alps, respectively).

When the cpp token PVP is set, ordering of the complex coefficients within the arrays is along diagonals of m (zonal wavenumber), and n (total wavenumber) space. Level index "k" (not to be confused with spectral truncation parameter "K") is the slowest varying index of these arrays. Spectral space computations on a vector computer are done most efficiently with this ordering scheme because it results in the longest possible vector lengths. Arrays ncoefi, nm, and nco2 in /comspe/ define lengths and starting addresses within the various spectral arrays. A pictorial representation of the structure of the spectral space arrays is presented in Figure 3.6.

When PVP is not set, spectral data ordering is changed in order to optimize performance on cache-based architectures, and to facilitate message-passing. The storage arrangement in this case is depicted in Figure 3.5

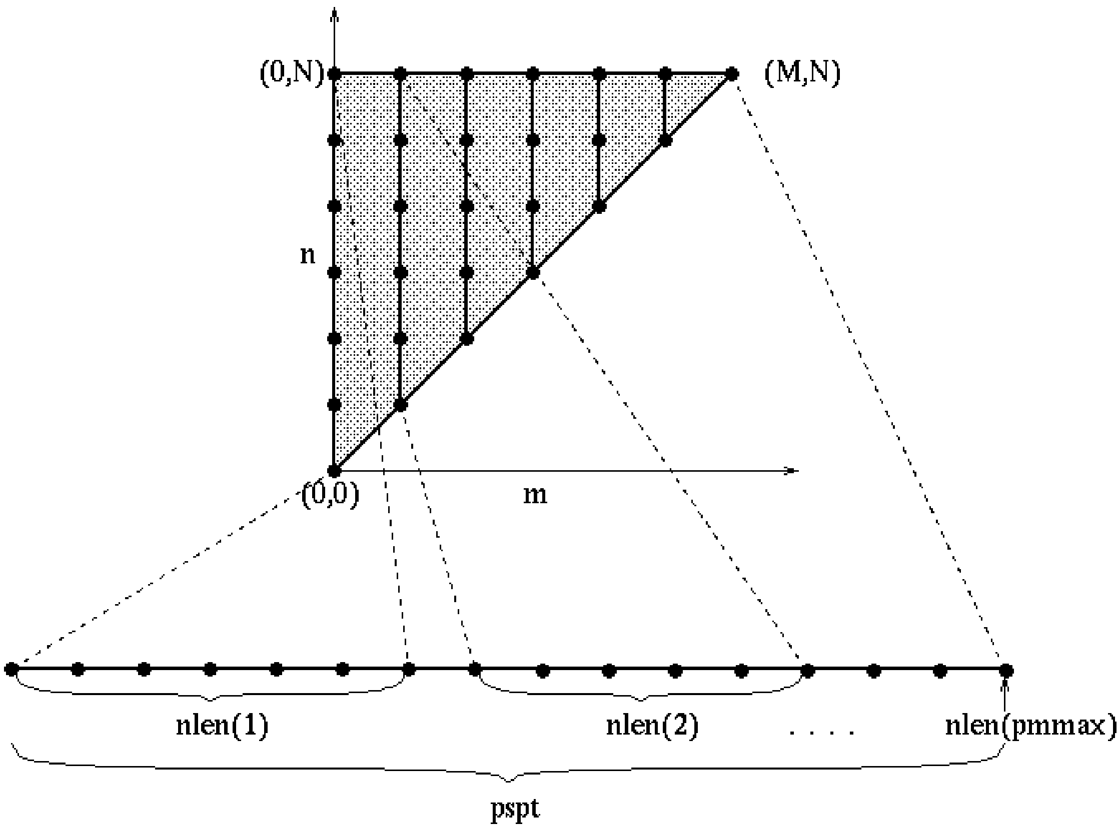
3.3.4 Other Common Blocks and Header Files: The Parameterization Interface

In keeping with a plug-compatible coding philosophy, parameterizations are designed to allow for simple modular replacement with other packages. The major consideration for such replacement is interfacing the data structures of the new code with those of the rest of the model. Each physics package has its own common block which contains only those physical constants which are required for its own use. For example, /comvd/ contains eight variables--just those constants that are required by the vertical diffusion package.

The various parameterizations that comprise the the model physics are hooked to the control code using a standard form of initialization and calling interface. Each parameterization contains an initialization routine called on start-up. In most cases, these routines are named "xxinti," where "xx" is the standard prefix for the particular parameterization (e.g. vdinti is the initializer for the vertical diffusion package). Most of these routines are called from subroutine inti, before the time integration procedure begins. These routines set constants in the parameterization-specific common blocks and do other initialization tasks that are not dependent upon time or latitude. Some data initialization is done within various block data subprograms.

Figure 3.5. Spectral storage arrangement when running on a parallel vector processor.

Dashed lines depict mapping of 2-dimensional conceptual location in triangular data structure to actual linear storage in memory.

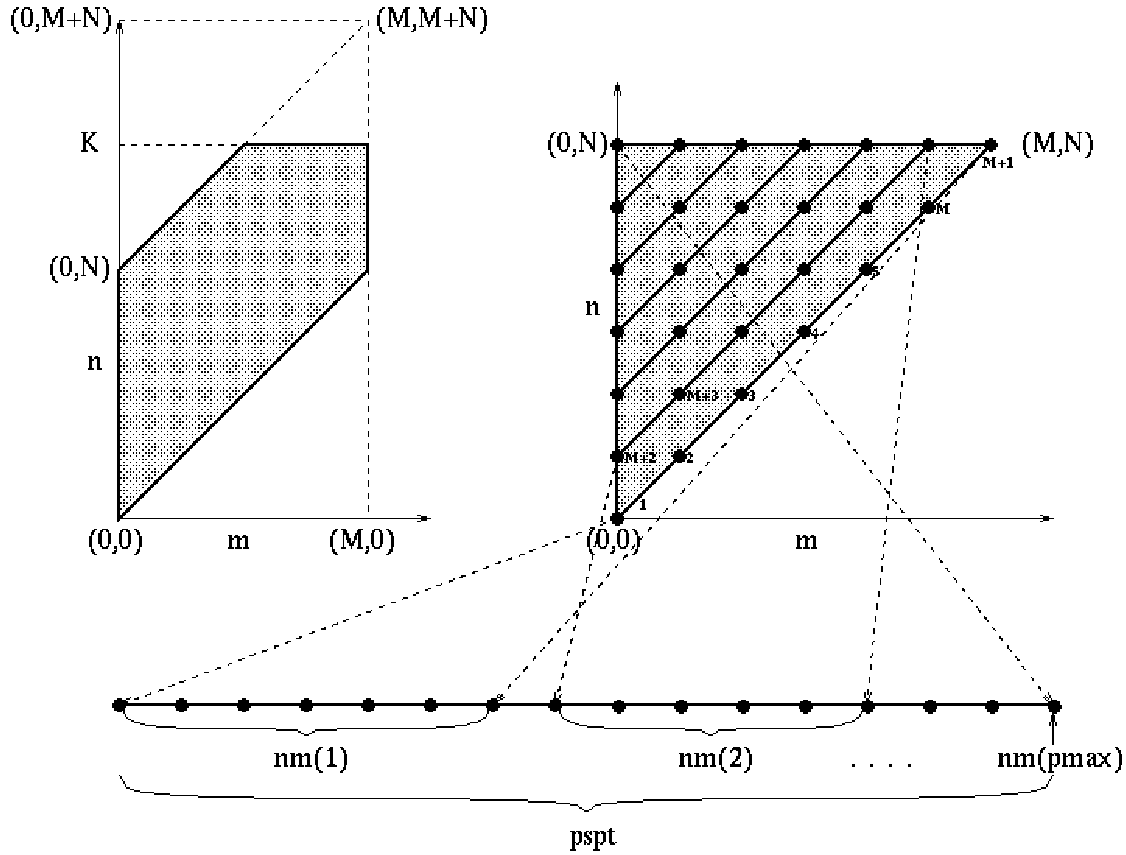


Certain constants may appear in more than one common block, as needed by various parameterizations. For instance, constant `cpair`, the specific heat of dry air, appears in `/comadj/`, `/comvd/`, `/comcon/`, and other common blocks. Variables in `/comcon/` are set in subroutine `initcom`. Then subroutine `inti` passes the required `/comcon/` constants into each "`xxinti`" routine, which sets the individual common block constants from the passed values. At this point unit conversions or other manipulations as required by the individual parameterization are performed.

Other common blocks (i.e. those not directly related to physics code) are defined by function and exist either to give their variables global scope or to avoid the necessity of passing certain variables through many argument lists.

Figure 3.6. Spectral storage arrangement when not running on a parallel vector processor.

Dashed lines depict mapping of 2-dimensional conceptual location in triangular data structure to actual linear storage in memory.



Each parameterization includes an interface routine that passes data between the high-level driving routines and the parameterization itself. This approach allows the parameterization to access each required array using a mnemonically named multidimensional array, rather than via the common block itself This is shown in the section "Grid-point Data Structures", "Model Buffers" .

3.4 Multitasking Strategy

CCM3.6 is designed to run most efficiently on shared-memory parallel vector machines. Details of the multitasking strategy are designed to be transparent to the user. Multitasking on a Cray PVP machine only requires that the user specify the number of processors to use at run time (by setting the environment variable $NCPUS).

The model can also be run multitasked on distributed memory architectures that support the MPI message-passing programming paradigm. The code should run on any machine or group of machines on which MPI is supported. The message-passing code has been validated on SUN (Solaris) and SGI (IRIX) machines. Hooks exist for T3D/E, RS6K, and HP (SPP-UX) machines. The T3D/E is no longer supported with the introduction of netCDF boundary data files. The IBM RS-6000 is no longer supported since NCAR no longer has RS-6000 machines available for testing purposes. The HP version is still experimental.

One design prerequisite is that identical simulations be guaranteed regardless of the number of processors utilized. This goal is accomplished by requiring that certain summations of critical quantities always be done within a single process (shared-memory multitasking), or done redundantly by each multitasked process (distributed memory multitasking). An example is the summation of the global mass integrals performed in `scan2`. Results from two runs executed with a different number of processors will be identical. When adding new code, the user should always ensure that the code is reproducible when running with one-processor versus multitasked.

3.4.1 Shared-Memory Multitasking

CCM3.6 has very little single-threaded work in it. One can therefore exploit all available parallel degrees of freedom (e.g., number of latitude bands) and see significant improvement in turn-around time compared to a single-threaded run. Thus, the number of processors to multitask over is limited to the number of latitudes. Observed scaling may be close to linear or significantly worse depending upon how well the memory system scales on the machine in question. For example, very good scaling has been observed on an SGI O2000 machine, but the same code doesn't scale nearly as well on the Cray PVP architectures. However, scaling may be dependent on the total amount of memory used. For example, a very high resolution run may not scale nearly as well as a lower resolution run.

The shared-memory code is multitasked using directives which take the form of Fortran comments. Such directives exist for Cray, SGI, and HP architectures. On the Cray, they take the form:

```
CMIC$ DO ALL SHARED (...) PRIVATE (...).
```

The `DO ALL` portion of the directive tells the compiler that each iteration of the next loop can be done independently. Names within the parentheses after `SHARED` indicate those variables that are global to the loop and will be shared across multiple processes. Conversely, variables declared as `PRIVATE` will have separate storage allocated for each process. Examples of variables that may be shared include those which are read-only within the loop, or which have separate storage already allocated for each iteration of the loop. Variables taking different values during separate iterations of the loop must be private. The loop index itself is an example of such a variable. Autotasking compiler directives exist only in routines `ccmoce, dyndrv, lsmdrv, scan1ac, scan1bc, scan2, scanslt, sltini, ccmoce,` and `somoce`. These routines drive all the physics and dynamics.

In any multitasked code, work done by one processor must be independent of the work being done (potentially) simultaneously on all other processors. Iterations of the Gaussian latitude loop in `scan1ac` are independent of latitude. Therefore, this routine contains a multitasked loop of the form:

```
CMIC$ DO ALL SHARED(...) PRIVATE(...)
      do lat=1,plat
      ...
      end do
```

Computations within the loop for each value of `lat` are done independently. Work is parceled out to available processors until all iterations are complete, with an implied synchronization point at the end of the loop.

In the SLT routines which drive multitasked loops (`scanslt` and `sltini`), each latitudinal iteration may be done in parallel. In the spectral dynamics driven by `dyndrv`, the Gaussian quadrature and semi-implicit timestep computations are parallelized over diagonals of "m-n" wavenumber space on PVP machines, and over total wavenumber "n" otherwise. Subroutine `dyndrv` also drives the horizontal diffusion calculations, which are parallelized over the vertical level index.

It is important to realize that there is no guarantee of the order in which multitasked iterations of a loop will be either initiated or completed. For this reason, coding constructs of the form,

```
    subroutine xxx(lat)
   if (lat.eq.1) then
   ... code to initialize static variables ...
   end if
```

will not work inside multitasked regions of code. If `lat=2` happens to be the first process to reach routine `xxx`, the variables set within the above "`if`" construct of the routine will not be properly initialized. To guarantee that routine `xxx` will work properly when multitasked, static variables which may vary across multiple calls to the routine (i.e. in this case across latitude bands) must be set in a single-threaded part of the code before the routine is called. Another result of the unpredictability of calculation order under multitasking is that direct access I/O to the work file units becomes necessary. If sequential access were used, there would be no guarantee that a given processor's request for a latitude band of data would fall in the correct order. Standard Fortran direct-access `read` and `write` statements are employed, so there is no loss of code portability.

Sequential I/O is used for the history files, even though the output order of latitude bands is totally unpredictable in a multitasked run. Sequential I/O is necessary because the history files contain variable-length records, and standard Fortran direct access I/O requires fixed-length records. To enable identification of the latitude band of each record by post-processing programs, the latitude index of each band is included as the first value in each data record of the history file.

3.4.1.1 Shared-Memory Management

A stack-based memory management scheme was crucial in designing a shared-memory multitasked version of CCM3.6. When a local variable is allocated on the stack, its lifetime is limited to the time the subroutine or function in which it is declared is active. Hence, stack-based local memory allocation makes for more efficient memory usage since memory is allocated to each routine only when required.

Multitasking is another important consideration in using stack-based memory allocation. Unlike static memory, stack memory declared within multitasked regions of code is automatically replicated across processes. Thus, use of the stack is the only reasonable way to ensure that multitasked routines are "reentrant," i.e., they may be entered by more than one processor at once without violating previously defined data space. For example, consider array `pmid` declared locally in subroutine `linemsbc`. Array `pmid` is dimensioned `(plond,plev)` with no latitude index because each instance of `linemsbc`, by virtue of its execution on a separate processor, has its own local copy of the array. The routine can therefore freely modify the contents of `pmid` without interfering with other concurrent instances of `linemsbc`.

There are a few places where locally defined memory is statically allocated within a multitasked region of the code. This is done for some read-only memory mainly to avoid having to reset certain variables every time a routine is invoked. These variables may be declared static via a Fortran **save** statement, or by virtue of appearing in a **data** statement. One disadvantage to running the model on multiple processors is that memory usage is increased as compared to a single-threaded run. The primary culprit is the stack memory replication across processors just discussed.

3.4.2 Distributed Memory Multitasking

The one-dimensional latitudinal data decomposition employed in the shared-memory multitasked code was also employed in the distributed- memory MPI implementation. Obviously this approach places limitations on the parallelism exploited in the application (i.e. a maximum of 64 processes at a horizontal resolution of T42), but it was felt to be the only viable approach to enabling either type of multitasking from within a single set of source code. One payoff of this approach is that nearly all routines which do gridpoint space computations are identical regardless of whether the target architecture is shared-memory or distributed-memory.

The spectral dynamics in the distributed-memory code is parallelized over Fourier wavenumber "m" rather than diagonals of "m-n" wavenumber space (see Figure 3.6) since the spectral transform technique requires summations over total wavenumber "n". These are best done on-processor (each processor will do the sum over "n" as these arrays are stored locally on the given processor). Since the loops which perform spectral space computations therefore look completely different in the distributed-memory code, some source files contain two subroutines (separated by *#ifdef*'s), one for each implementation. Shared-memory code was not rewritten to employ this same data ordering (which would have eliminated a large number of *#ifdef*'s) because much longer inner loops are generated when the data are stored along diagonals of "m-n" wavenumber space (resulting in much higher vector performance on a PVP machine).

Due to hemispheric symmetry considerations in the spectral dynamics, the number of processors utilized in the distributed-memory code (`$NPES`) must be an even number. Load balance considerations dictate that a value of `$NPES` which divides evenly into the total number of Gaussian latitudes works best, though this is not mandatory.

3.4.2.1 Distributed Memory Management

Storage for arrays which are global (i.e. contain full latitudinal dimensionality) in the shared-memory code is allocated off the heap in the distributed-memory code, instead of a static declaration using Fortran *parameter*s. Reasons are: 1) to avoid wasting storage, and 2) to enable a single executable binary to be run on an arbitrary number of processors without the necessity of recompiling. Storage is allocated using system routine `hpalloc` on the Cray or `malloc` on other architectures, and accessed using Cray pointers. Gridpoint data structures are defined only for those latitude bands which reside on-processor. Likewise, spectral data structures are only defined for Fourier wavenumbers which are on-processor.

Storage for gridpoint arrays which are not global in the shared-memory code is allocated exactly the same way in the distributed-memory code. Namely, either statically or stack-based as appropriate using Fortran *parameters* .

3.4.2.2 Distributed Memory I/O

All I/O in the distributed-memory code is done by a master processor. This includes only the history and restart files, since the ability to cycle the various model buffers to an out-of-core device was not enabled. Unfortunately this radically different approach to I/O manifests itself in radically different coding constructs, and a large number of *#ifdef* constructs in the routines which do I/O. Fortunately, most users need not be concerned with routines which write the history and restart files.

3.5 Output Datasets

CCM3 generates several output datasets during execution. The purpose of this section is to describe the structure and contents of the output history files, and the restart files, as well as provide an indication of where and how they are written in the code. Knowledge of the material presented in Sec. 2.2 through 2.5 is assumed.

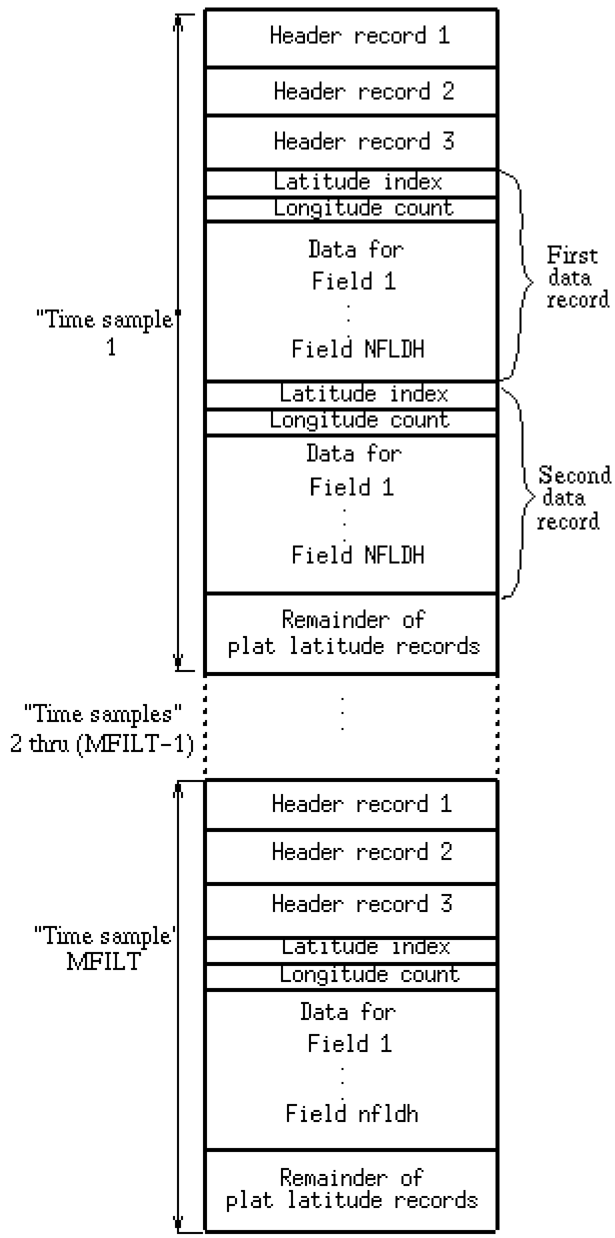
3.5.1 History Files

History files contain descriptive information and data values written at specified times during the run. As discussed in Sec. 2.5 , the model automatically writes a "primary" history file, and has the capability of writing up to five "auxiliary" history files. A "Master Field List" (Table 2.13) contains a list of all the variables that can be output to a history file. Certain fields on the Master Field List appear by default on the primary history file (as indicated in Table 2.13). Using the `CCMEXP` namelist variables `PRIMARY` , `EXCLUDE` , and `AUXF` , the contents of each history file can be customized. Other namelist variables control the time frequency at which each history file is written, along with the type of field averaging (averaged, instantaneous, minimum, maximum, or monthly-averaged), packing density, and number of time samples written to a single history file. The NCAR Mass Store System can be used to archive history files. History files are written directly in binary format rather than netCDF due to performance considerations and lack of packing support. However, we support a conversion tool (ccm2nc) that converts CCM history files into netCDF.

3.5.1.1 History File Format

The structure of a history file is illustrated in Figure 3.7. One history file contains up to `MFILT` time samples, with each time sample consisting of three header records followed by `plat` latitude data records (`plat` is a Fortran parameter set to the number of latitudes for a model run). A latitude data record is composed of a latitude index, a longitude count, and all data fields with the given latitude index. Details of the latitude data records are discussed in "Latitude Data Records" . History files are written to the disk sequentially and in binary format.

Figure 3.7. History file format.



3.5.1.2 History File Header Records

The first three records in a time sample comprise the history file header. These records are written from *common* blocks `/comhdi/`, `/comhdc/`, and `/comhdr/` (declared in the file `comhed.h`). Using the three-record header structure, the model splits the integer, character, and real header values into separate records, making the code more easily portable to other computers. The header of an output history file serves two functions:

- It provides the information required by postprocessors to read and locate specific data fields.

- It serves an archival function, storing information such as date/time and job sequence numbers concerning the Mass Store tape on which it is written.

Contents of the three header records are shown below in Table 3.1 (integer), Table 3.2 (character), and Table 3.3 (real). Each variable is listed by name and word number within each header record, along with a short description and a typical value based on the T42, 18 vertical-layer case. Output fields in a CCM3 history file are located on the hybrid vertical coordinate grid. Thus, three arrays of hybrid coefficients are saved in the real header record along with the Gaussian latitudes and weights (see Table 3.3).

Table 3.1: History File Header Record 1-- Integer Variables

Variable Name	Word Number	Description	T42 Value
LENHDI	1	Length of header record 1	292
MFTYP	2	Format code for this history file	43
MFILH	3	Number of this "time sample" within this disk file or Mass Store tape.	1
MFILTH	4	Maximum number of time samples in this history file.	20
NRBD	5	Number of records before data records	3
MAXSIZ	6	Length of packed data record in words	33662
NDAVU	7	Length of the data record in words after unpacking	65282
	8	Unused	0
NLON	9	Number of longitude points in one latitude line	128
NLONW	10	Number of longitude data values written on the history file	128
NOREC	11	Number of latitude lines and number of data records	64
NLEV	12	Number of vertical levels written on the history file	18
NTRM	13	M spectral truncation parameter	42
NTRN	14	N spectral truncation parameter	42

NTRK	15	K spectral truncation parameter	42
NFLDH	16	Number of fields in this time sample	85
NSTEPH	17	Timestep on which this data was written	0
NSTPRH	18	Iteration number for the start of this run	0
NITSLF	19	Number of timesteps since last tune sample was written	0
NDBASE	20	Base day number for this case	0
NSBASE	21	Base number of seconds for this case	0
NDCUR	22	Current day number corresponding to NSTEPH	0
NSCUR	23	Seconds of the current day NDCUR	0
NBDATE	24	Base date (yr mo day) as 6-digit integer	000901
NBSEC	25	Seconds to complete NBDATE date. (**Note**: may be different from NSBASE.)	0
NCDATE	26	Current date (yr mo day) as 6-digit integer corresponding to NSTEPH	000901
NCSEC	27	Current seconds for date NCDATE	0
MDT	28	Model timestep in seconds	1200
MHISF	29	Frequency in timesteps at which time samples are written	72
MFSTRT	30	Flag to indicate type of run: = 0 for initial run = 1 for restart = 3 for branch run	0
LENHDC	31	Length of header record 2	259
LENHDR	32	Length of header record 3	239
MPSIG	33	Position in header record 3 of first word of sigma value list. sigapb(1) = REALHD(MPSIG)	1

MPLAT	34	Position in header record 3 of list of latitude lines. `hdlat(1) = REALHD(MPlat)`	112
MPWTS	35	Position in header record 3 of list of Gaussian weights. `hdwt(1) = REALHD(MPwts)`	176
MPFLDS	36	Position of header field information list in header record 1 (integer values)	38
MPCFLD	37	Position of field information list in header record 2 (character values), given as a number of 8-character items from the start of record 2	90
MFLDS	38	Integer field information, dimensioned `(3, NFLDH)`. See description of `MFLDS` array in Table 3.4 .	--

Table 3.2: History File Header Record 2 -- Character Variables*

Variable Name	Word Number	Format	Description	Example T42 Value
MCASE	1	C*8	Case identifier	clsst01
MCSTIT	2-11	C*80	Case title	Test run
Current History File				
LNHSTC	12-21	C*80	Pathname of this history file	*/USER/csm/clsst01/ccm3/hist/h0001*
LDHSTC	22	C*8	Date this header record was written (MM/DD/YY)	04/27/98
LTHSTC	23	C*8	Time this header record was written (HH:MM:SS)	00:28:25
LSHSTC	24	C*8	Unused	
First History File of Case				
LNHSTF	25-34	C*80	Pathname of first history file for	*/USER/csm/clsst01/ccm*

			this case	*3/hist/h0001*
`LDHSTF`	35	C*8	Date this case started (MM/DD/YY)	04/27/98
`LTHSTF`	36	C*8	Time this case started (HH:MM:SS)	00:28:22
`LSHSTF`	37	C*8	Unused	
Initial Dataset File				
`LNHSTI`	38-47	C*80	Pathname of initial dataset file	`SEP1.T42.0198.nc`
`LDHSTI`	48	C*8	Date initial dataset was created (MM/DD/YYun)	02/06/98
`LTHSTI`	49	C*8	Time initial dataset created (HH:MM:SS)	06:26:35
`LSHSTI`	50	C*8	Unused	
`--`	51-63	--	Words 51-63 are unused.	
Time-variant SST Dataset File				
`LNHSTVS`	64-73	C*80	Pathname of SST boundary dataset	`STR-sst.T42.0596.nc`
`LDHSTVS`	74	C*8	Date SST boundary dataset was created (MM/DD/YY)	05/17/97
`LTHSTVS`	75	C*8	Time SST boundary dataset was created (HH:MM:SS)	17:03:13
`LSHSTVS`	76	C*8	Unused	
Ozone Dataset				
`LNHSTVO`	77-86	C*80	Pathname of ozone boundary dataset	`ozn.0596.r8.nc`
`LDHSTVO`	87	C*8	Date ozone boundary dataset was created (MM/DD/YY)	09/21/92

LTHSTVO09/21/92	88	C*8	Time ozone boundary dataset was created (HH:MM:SS)	13:34:26
LSHSTVO	89	C*8	Unused	
MCFLDS	90	C*8	Character field information, dimensioned (2,NFLDH). See description of MCFLDS array in Table 3.4 .	--
**All character strings are left-justified, blank-filled, and multiples of 8 characters.*				

Table 3.3: History File Header Record 3 -- Real Variables

Variable Name*	Word Number	Description
REALHD(MPSIG)		Contains 3 arrays of (2 * NLEV +1) hybrid coefficients as follows: sigapb = hybrid A+B coefficients siga = hybrid A (pressure) coefficients sigb = hybrid B (sigma) coefficients
REALHD(MPLAT)		Points to the first word, relative to the start of the header, of a list of NOREC Gaussian latitudes in degrees (south to north)
REALHD(MPWTS)		Points to the first word of a list of NOREC Gaussian weights corresponding to the Gaussian latitudes
*MPSIG, MPLAT, *and* MPWTS *are from header record 1, words 33, 34, and 35, respectively.*		

Two arrays mflds (starting at word 38 of the integer header) and mcflds (starting at word 90 of the character header) contain integer and character information describing each field stored in a latitude data record. Table 3.4 shows details of the contents of these two arrays.

Table 3.4: Description of Field Information List Arrays, MFLDS and MCFLDS

Variable Name	Description

`MFLDS(1,IFld)`	Flag for number of levels and field flag, packed in one integer word. The ones digit is for the level number flag; the tens digit states whether the field is an instantaneous value, a value averaged between history file writes, a time minimum between writes, or a time maximum between writes. ones digit of `MFLDS(1,ifld)` 0 = single-level field, 1 = multilevel field at layer interfaces, 2 = multilevel field at layer midpoints. tens digit: 0 = instantaneous field, 1 = time-averaged field. 2 = minimum field value since the last time sample. 3 = maximum field value since the last time sample.
`MFLDS(2,IFld)`	Pointer to the first field value in packed data record.
`MFLDS(3,IFld)`	Data-packing flag (set to 1 if not packed).
`MCFLDS(1,IFld)`	Field name for field number `IFld`. (8 characters)
`MCFLDS(2,IFld)`	Units of this field (SI units). See `Table 4.1 .` (8 characters)
The index `ifld` goes from 1 to `nfldh` (the number of fields in the history buffer).	

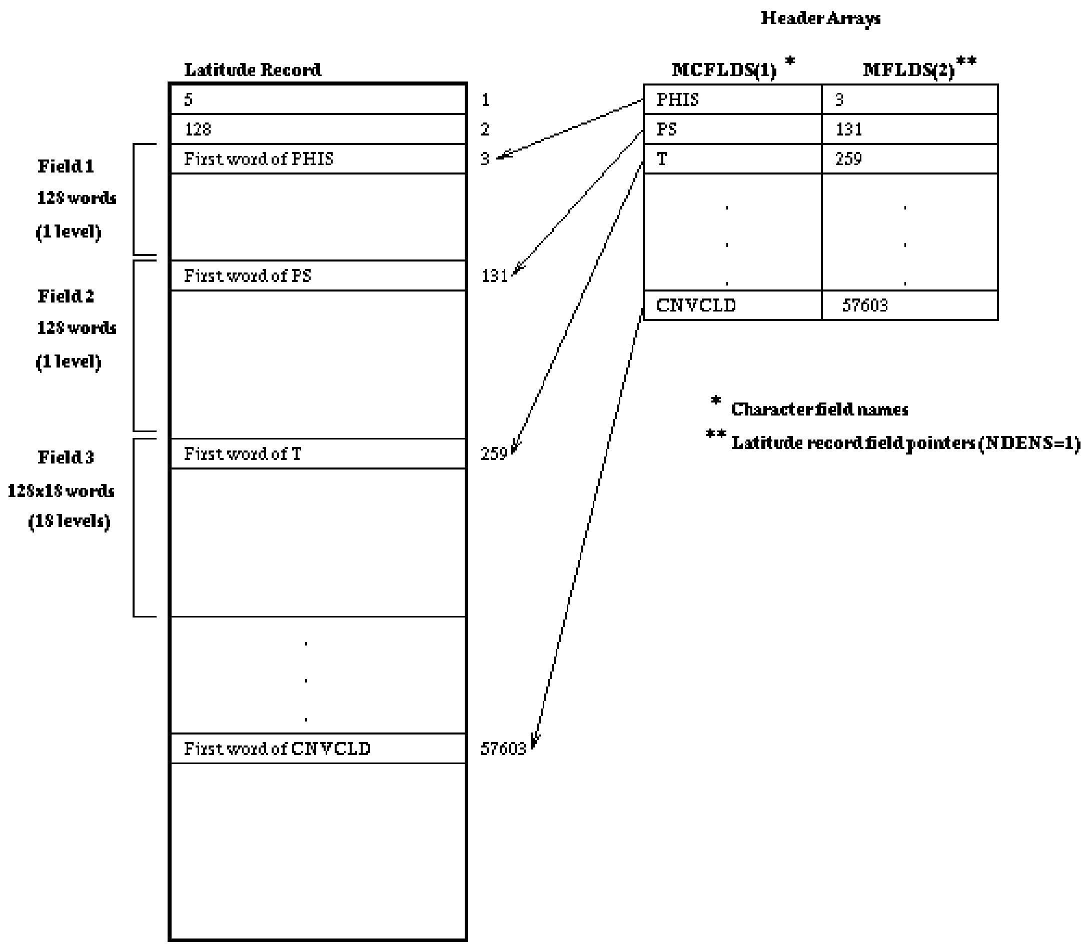
3.5.1.3 Latitude Data Records

Following the header records in each time sample are the latitude data records. All words in a latitude data record are floating point. When the model is multitasked the latitude records are written in a random order. This occurs because writing of the latitude data records into a sequential access disk file occurs from within a multitasked loop over latitudes (in `linemsbc`). So a given latitude is written when the processor that calculates that latitude is finished.

Figure 3.8 illustrates a typical latitude record of a time sample within a history file. In this figure, latitude record number 5 (the fifth latitude from the southern-most) is depicted for T42 horizontal resolution. This figure also shows how the pointers in `MFLDS` relate to the field data contained in the record, along with the variable names in `mcflds`.

Figure 3.8. A Typical history file latitude record.

Showing unpacked data, and its relationship to the pointers in MFLDS and variable names in MCFLDS.



A few important items to notice about the latitude data record are listed below:

- The first word of the record is an index into the latitude array in the real header record, identifying the latitude associated with this data. This field is required due to the random ordering of the latitude data records within a time sample.
- The second word of the record is a count of the number of longitudes contained in that record. Currently, this number is always *float(*`plon`*)* (the latitude record contains only floating point values) but allows for the possible future implementation of a variable number of longitude grid points along each latitude line.
- Field information starts in word 3 of the record. The pointers contained in the header array `MFLDS` point to the beginning word of data for each field, packed or unpacked. Thus, it is possible to read in a packed record and unpack only the desired fields.

- All fields in a latitude record correspond to a single latitude index. If a field is single level, then it contains grid-point data at `plon` longitude points, for the given latitude index. If a field is multi-level, then it contains grid-point data at (`plon` x `plev`) points, for the given latitude index.

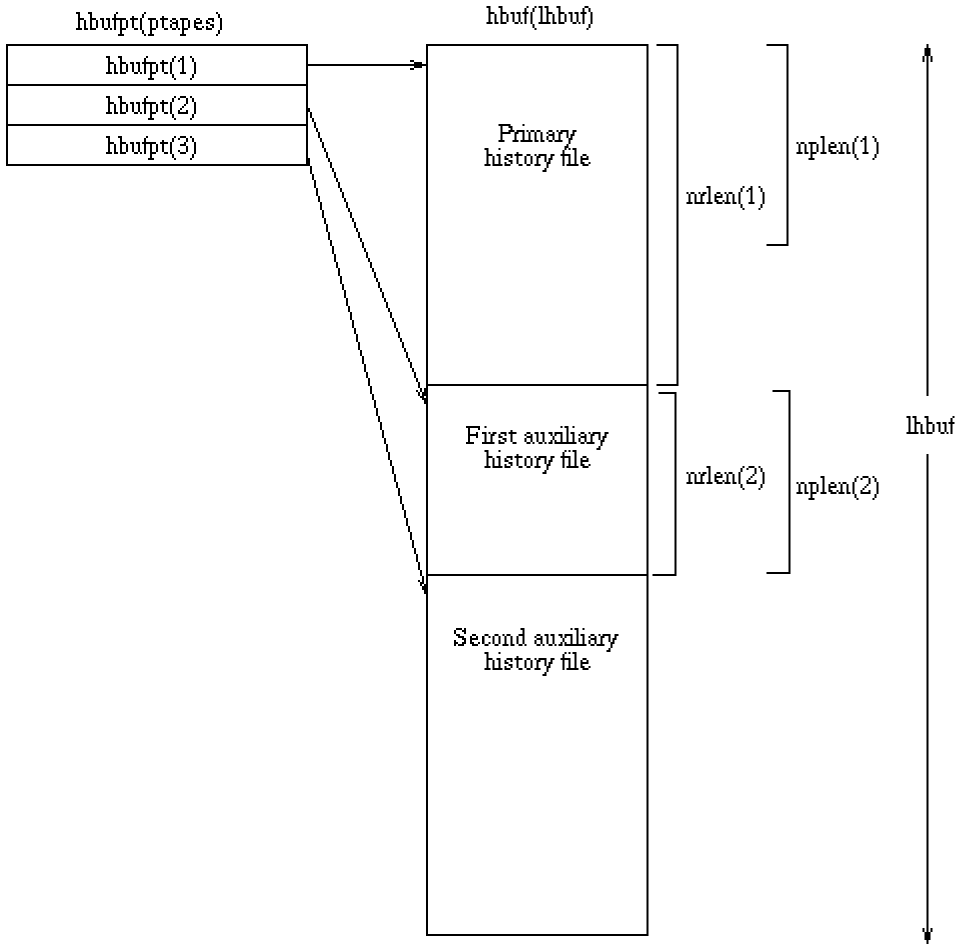
3.5.1.4 History Buffer

The history buffer is dynamically allocated storage space that is used to accumulate history file data fields. As described in Sec. 2.2, several namelist variables are used to define the fields written to each history file (primary and auxiliary). Recall that the variable `NHTFRQ(i)` indicates when each time sample will be written, and `NINAVG(i)` tells what type of field should be output (averaged, instantaneous, etc.). Since many iterations may pass before time samples are output to a history file, it is necessary to have storage space available in which the history file fields can be accumulated and stored. Namelist variables `PRIMARY`, `EXCLUDE`, and `AUXF` permit the user to specify which fields to save in the primary history file and which, if any, to save in the auxiliary history files. The model uses this information along with the default fields on the Master Field List (see Table II.13) to determine the history buffer length (set to the variable `lhbuf)` required to store the all the history fields. The model then dynamically declares the memory needed to hold the history buffer.

Figure 3.9 depicts the history buffer for a case in which the user has requested two auxiliary history files in addition to the primary history file. The history buffer is represented by the vertical rectangle in the center of the figure. Notice that the buffer is divided into three sections, the first corresponding to the portion of memory devoted to fields on the primary history file, followed by the portions allotted to the first and second auxiliary files, respectively. The values in array `hbufpt` point to the starting word of the latitude data record for each history file in the buffer. Arrays `nplen` and `nrlen` contain the packed and unpacked field lengths of each data record. Recall namelist variable `NDENS(i)` specifies the packing density of fields in each history file. For the buffer shown in Figure 3.9, `NDENS(1)=2`, thus `nplen(1)` (the packed length) is one-half the length of `nrlen(1)` (the unpacked length) as shown.

Figure 3.9. History buffer layout.

The length of history buffer, hbuf, is specified by variable lhbuf. The starting location of each history file latitude record in the buffer is pointed to by the hbufpt array. Lengths of the packed and unpacked latitude data records for the primary and first auxiliary history files are shown by nplen and nrlen, respectively.



Allocation of memory for the history buffer occurs during the initialization phase of a run, in subroutine `intht`. If the buffer is contained entirely in-core, then the model calls routine `getmem` to dynamically allocate a segment of in-core memory of length `plat` x `lhbuf`. The storage address of the first word in this in-core memory block is pointed to by the Cray pointer `phbufic` and its associated two-dimensional pointee array `hbufic(lhbuf,plat)`. After being allocated, the in-core history buffer exists and is accessible for the remainder of the run.

During the time integration phase, field values to be output to a history file are accumulated in the history buffer. For each field that will be saved in the history buffer, a call is made to routine `outfld`. Calls to `outfld` occur within a multitasked loop over latitudes, from routines `linemsbc` or `linemsac` and below. When running multitasked, each process will perform calls to `outfld` and load the history buffer with fields from a single latitude index. Access to the section of the in-core history buffer relevant to that latitude index is facilitated by the pointer array `hbuf`. The following block of code, extracted from `linemsbc`, shows how this is accomplished.

```
subroutine linemsbc(lat   ,tdt   ,ps   ,psm1  ,u3  ,
 .
 .
 .
pointer (phbuf,hbuf)
real hbuf(lhbuf)          ! history buffer pointee array
 .
 .
 .
 if (incorhst) then
   phbuf = loc(hbufic(1,lat))   !in-core history buffer
 else
  .
  .
  .
 call outfld('PS      ',psm1,plond,lat,hbuf)
```

In this code, `lat` refers to the particular latitude index that is being calculated, and is passed as an argument to `linemsbc` from a multitasked loop in `scan1bc`. The pointee array `hbuf` is associated with the portion of `hbufic` that corresponds to `lat` (since `INCORHST` is true when the buffer is in-core). The call to `outfld` that is shown will load the surface pressure array, `psm1`, into the history buffer. A similar call to `outfld` exists for every history file field.

If the history buffer is declared out-of-core, the model will use the SSD to accumulate history file fields. If no SSD exists, then the model will use a local disk file to store the out-of-core history buffer. This should be avoided since disk I/O is typically very slow. As described in "Out-of-Core Data Storage: The SSD Work Units" , the SSD is high-speed, out-of-core secondary storage that is accessed using Fortran *read* and *write* statements. Thus, the out-of-core history buffer can be thought of as a direct access Fortran file with extremely fast I/O. The SSD file contains `plat` data records, each of length `lhbuf`. When the model needs to accumulate fields in the history buffer (from within multitasked routines `linemsbc` and `linemsac`), the following steps are executed on each processor:

- In-core memory of length `lhbuf` is allocated from the heap using routine `getmem`.
- The portion of the history buffer that corresponds to the specific latitude index being processed is read in from the SSD using routine `readric`, and saved in the newly allocated in-core memory. The pointee array `hbuf` becomes associated with the newly allocated space for the history buffer fields.
- History fields are accumulated in array `hbuf` through calls to routine `outfld`.
- Prior to completion of the multitasked routine, `hbuf` is written to the out-of-core history buffer located on the SSD using routine `writeric`. In-core space that was allocated for

`hbuf` is then released by calling `freemem`.

The following block of code shows how the out-of-core history buffer is accessed in `linemsbc`. This differs from the previous block of code shown, in that, it includes code to access the out-of-core history buffer from the SSD.

```
subroutine linemsbc(lat   ,tdt   ,ps   ,psm1  ,u3  ,
 .
 .
 .
pointer (phbuf,hbuf)
real hbuf(lhbuf)          ! history buffer pointee array
 .
 .
 .
if (incorhst) then
  phbuf = loc(hbufic(1,lat))   !in-core history buffer
else
  call getmem('LINEMSBC',lhbuf,phbuf) !out-of-core hbuf
  call readric(sunit,hbuf,lhbuf,lat)
endif
 .
 .
 .
call outfld('PS      ',psm1,plond,lat,hbuf)
 .
 .
 .
if (.not.incorhst) then
  call writeric(sunit,hbuf,lhbuf,lat) ! write to SSD
  call freemem(phbuf)
endif
.
.
.
return
```

Again, the current latitude index `lat` is passed as an argument to `linemsbc` from `scan1bc`. The pointee array `hbuf` and its pointer `phbuf` are then declared. Because `INCORHST` is false when the history buffer is kept out-of-core, the model calls `getmem` and `readric` to access the history buffer as described in the steps above. Just as with the in-core history buffer, fields are accumulated in array `hbuf` via calls to `outfld`, as shown for the surface pressure array `psm1`. After all history fields have been loaded into the `hbuf` array, and prior to executing the *return* statement in `linemsbc`, the history buffer fields are written to the out-of-core history buffer on the SSD by calling routine `writeric`. Once this has been completed, `freemem` is called to release the in-core memory and return it to the heap.

3.5.1.5 History File Code Flow

Many routines are involved in the initialization and writing of history files. Since it is difficult to track these routines within the code, a brief summary of the code flow pertaining to history files is given below. Refer to the calling tree in Appendix B to view the relative locations of the routines cited below.

Initialization of the history files occurs in several routines. Namelist variables associated with the history file controls are initialized in routine `preset` and read into the model in `data`. If a restart run is being conducted, existing history files that are not full (contain less than `MFILT` time samples) are opened and repositioned in routine `restrt`. Routine `intht` is the main driver routine that initializes new history files and determines the contents and size of the history buffer. This routine first calls `bldfld` to build the Master Field List (see Table 2.13) and `fldlst` to construct the history buffer using the Master Field List and the values of the namelist variables `EXCLUDE`, `PRIMARY` and `AUXF`. `inthed` then initializes the variables in the history file header common to all history files and `intht` dynamically allocates memory for the in-core history buffer or opens the SSD file necessary for the out-of-core history buffer. If any history buffer regeneration files exist (as may happen in a restart run) then `splitg` is called to read its contents into the in-core history buffer or the out-of-core SSD file. Finally, `rgnhbf` initializes the history buffer regeneration files (if needed).

Writing of the history fields to the buffer and writing of the history buffer contents to a history file, occurs from within the time integration loop of `stepon`. History fields are accumulated in the history buffer every timestep. However, as described in Sec. 2.2, a time sample is only written to the disk history file as directed by namelist variable `NHTFRQ`. The model sets logical variable `hstwr` to true if it is time to write a time sample to a given history file. Of the subroutines called by `stepon`, only `scan1bc`, `scan1ac`, and `wrapup` directly interact with the history buffer or files. Routine `wrapup` closes any full history files and archives them to the NCAR Mass Storage System if applicable.

Several common blocks are associated with the history files. These common blocks are defined in the following header files: `comhst.h`, `comhed.h`, `commss.h`, `comlun.h` and `comloc.h`.

3.5.2 Restart Files

As explained in "Restart Datasets", the restart files contain information required to either continue a run from its termination point or to begin a branch run. A single restart pointer file and up to four types of restart files may be written during a model run. The restart pointer file along with the master and primary restart files are always written (if `NREFRQ` is nonzero), while secondary and history buffer restart files are only written if necessary. Restart files are written as sequential access binary files. Since, they are only intended to be written or read by the model, no facility exists with which to conveniently view them.

The restart pointer file is an ASCII text file containing the Mass Store pathname of the most recently written master restart dataset. Thus, for a "restart run" (where the namelist variable

NSREST equals 1), the model will read the restart pointer file, acquire the appropriate restart files, and commence running the model from the end point of the previous run. The restart pointer file is only written to disk. Its full pathname can be specified by the namelist variable REST_PFILE. If this namelist variable is not specified, restart pointer files will be written to /HOMEDIR/ccm3.CASEID.rpointer, where HOMEDIR is the users home directory (environment variable $HOME) and CASEID is the case name for the run as specified by the namelist variable, CASEID. The restart pointer file is written from routine wrapup.

The master restart file contains information necessary to restart the model. Variables saved in this file include the current timestep of a restart run (nstepr), the history buffer length, model date and time, history file specifications, and primary and secondary restart file specifications. The master restart file is written from routine wsds, which is in turn called by wrapup.

The primary restart file contains data from the main model buffers and 3-dimensional arrays. This allows the model to set all variables in a continuation run as they were at the time the restart datasets were written. Data from the following common blocks are saved on the primary restart file: /coreioab/, /com3d/, /comslt/, and /comsrf/ (see the header files of the same name for details of these *common* blocks). The primary restart file is written from routine wrtrs1, which is called by scan1bc.

A secondary restart file is written if the absorptivities/emissivities will not be calculated on the first timestep after restart, and therefore must be saved on a restart dataset. For a stand-alone run this occurs if the primary history file write frequency, NHTFRQ(1), is not a multiple of the absorptivity/emissivity calculation frequency, IRADAE (note that for a CSM flux coupled run, only the flux coupler determines when the restart files are written). It is advisable to avoid this situation if possible, since this dataset is relatively large even for the standard T42 model. On high-resolution runs, the model will split the secondary restart file over latitudes if necessary. This file is written from routine wrtrs1.

A history buffer restart file is written in order to retain the accumulated values in the history buffer if restart files are to be written on a timestep when one or more history file time samples are not written. A separate restart dataset is written for each such history file (primary and/or auxiliary). Each history buffer restart file contains the portion of the history buffer pertaining to that history file. If necessary, the model will split the history buffer restart file over latitudes. History buffer restart files are also written from routine wrtrs1.

Writing of the restart files occurs from within the time integration loop of routine stepon . Restart files are not written every timestep. The logical variable rstwr is used to determine if it is time to write the restart files. No restart data is written on timestep zero. Routine wrapup completes the task of writing the restart files begun in scan1bc . Common blocks associated with restart files are defined in the header files comlun.h and comloc.h .

3.6 Disk and Mass Store File Management

File management includes all activities related to the creation, use, and maintenance of sequential and direct-access files and includes acquiring input files, accessing data from these files, creating output files on disk, and archiving them if applicable.

Several input datasets are required to begin a new case or continue a previous one (see "Model Input Datasets" and "Restart Datasets"). Often these datasets contain a complete MSS path (e.g. `/XXX/yyy/zzz/filename`). As a first attempt to find the dataset, the model strips away the MSS part of the filepath and checks if `filename` exists in the current directory. If the file is not found, the path specified by namelist variable `DATADIR` is then prepended to the filename and is checked as a full pathname. If the file is still not found, the full pathname of the file as specified in the namelist input is searched on local disk. Finally, if this search is not successful, a subroutine call or shell command will be invoked to retrieve the file from the MSS. Users running on a machine not connected to the NCAR MSS can build the appropriate archival command for their particular site by modifying the few routines containing `#ifdef SHELL_MSS` CPP directives.

Model output files (see "Model Output Datasets") are created in the local directory. Depending on the model's horizontal and vertical resolution, the frequency at which output files are created and the size of the local file system, the aggregate amount of output data created could potentially overflow local disk space. Three control mechanisms exist: (1) control the rate at which data are generated; (2) control the size of the output datasets; and (3) archive the data to an off-line device. Control over the rate at which history and restart data are generated is provided via the namelist variables `NHTFRQ` and `NINAVG`. Control over the size of the output datasets is achieved by limiting the number of fields and/or compressing the data before it is written to local disk. Namelist variables `EXCLUDE` and `NDENS` provide this functionality. Once an output file is written to the local disk, it may be archived to an offline storage device. If running on an NCAR Cray, a Fortran interface to the local mass storage system is invoked. Otherwise, the cpp token `SHELL_MSS` may be defined, in which case a shell command is built to do the same job. NCAR archival commands are "mswrite" and "msread". Users at other locations will need to modify the code to build the appropriate command for their local environment. If running on an NCAR Cray, output datasets are removed from local disk after they are archived to the MSS, with the with the exception of the final set of history and restart files.

3.6.1 Disk File Management

Fortran logical unit numbers are determined at execution time. The model keeps a catalog of these numbers and provides user access through two interface routines. `navu` returns the next available (i.e. unused) Fortran unit, and `relunit` releases a unit number back to the model for later use.

Disk file systems can become very fragmented. Large files such as history and restart files are slow to read or write when this happens. Consequently, for large datasets the model pre-allocates space equal to the expected length of the output file using the **setf** command with the **-c** option. Routine `prealc` invokes **setf** to find as much contiguous space as possible for large output file.

3.6.2 Use of the NCAR Mass Storage System

When run on any of the NCAR computer systems, CCM3.6 uses the NCAR Mass Store System for archival storage of output datasets. Input and output datasets are copied between local disk and the MSS and accessed either via Fortran-callable routines, or via a shell command. Data are staged from the MSS to disk via **msread** (available both as a Fortran routine or a shell command). These requests are issued as synchronous operations. Hence, the model will wait until an **msread** is completed before resuming execution. However, **mswrite** commands (which stage data from disk to the MSS) are normally issued asynchronously for efficiency reasons. Asynchronous writes to the MSS are made by calling NCAR-specific routine **mswrite** with the **nowait** option. This allows the time integration to continue without having to wait for the file to reach the Mass Store.

3.6.3 Issuing Shell Commands from the Model

File operations that cannot be handled with Fortran library calls are accomplished by issuing shell commands from within the model code. This is done via **ishell** on Cray machines and **system** on other machines. The number of these calls is minimized by making use of the Fortran interface to commands such as **mswrite**. However, no such interface routine exists when disk space is pre-allocated on Cray PVP machines. This is done with the **setf** command, which has no Fortran interface.

4. Changing the Model

The most common changes to the model are adding new variables, adding transported constituents, changing the content of the history file, changing the model resolution, and adding a parameterization. This section provides some guidelines for making these kinds of changes.

4.1 Modifying the Code

If the user wishes to include modified code, a disk directory should first be created where all the modified code will reside. The file, `Filepath`, should have this new directory as its first entry (`Filepath` is created by the build script and therefore, should be modified in the script). **gnumake** will first look in this user directory for files that need to be compiled. Finally, the user should run the build script to create or update the model executable.

4.2 Adding New Variables

If the user wishes to adding new variables to the model, such as adding variables to a history file, adding new diagnostic fields adding transported constituents (Sec. 4.3), or adding variables within a physical parameterization (Sec. 4.5), two different cases need to be considered.

Case 1: **The variable already exists and the user wants to save it on a history file.**

If the field is in the Master Field List (see Table 2.13), the user must modify the code by uncommenting the `outfld` call for that field and must also add it to the history output via namelist variables `PRIMARY` or `AUXF`. `PRIMARY` will place the field on the primary history file, while `AUXF` will place the field on an auxiliary file, as described in Table 2.5.

If the field is not on the Master Field List, the user must add it to the list by modifying `bldfld`. Within `bldfld`, fields are added to the Master Field List by calling `addfld`. Four pieces of information are passed to `addfld` in an argument list:

1. Field name:
   8-character field name, left-justified, alphanumeric or spaces only.

2. Field units:
   8-character units description. See Table 4.1

3. Field level indicator:
   `nsingle` = single-level
   `nmulth` = multi-level at layer interfaces

`nmultf` = multi-level at layer mid-levels

4. Field active/inactive flag:
   A value of `active` indicates that this field will appear on the primary history file by default, whereas `inactive` indicates it will not. These flags can be overridden by namelist variables `PRIMARY` or `EXCLUDE`.

Table 4.1: SI Units of History File Fields

Abbreviation	Description
M	meter
KG	kilogram
S	second
K	degree Kelvin
MOL	mole
N	Newton
PA	Pascal
J	Joule
W	Watt

An example extracted from `bldfld` is shown below: These three calls are responsible for including `PHIS`, `PS`, and `T` as active (default) fields on the Master Field List. Notice that the arguments passed to `addfld` are the 8-character field name, the 8-character field units, `nf` (a field counter used by `bldfld`), the level indicator, and the active/inactive flag.

```
call addfld('PHIS    ','M2/S2   ',nf,nsingl,active)
call addfld('PS      ','PA      ',nf,nsingl,active)
call addfld('T       ','K       ',nf,nmultf,active)
```

The user must then add an `outfld` call for the field at the proper location in the code. This means that `outfld` must be called either from `linemsbc`, `linemsac`, or from a routine below either one in the calling tree. For example, the `outfld` call for the field `T`, taken from `linemsbc`, is shown below:

```
call outfld('T       ',t3m1,plond,lat,hbuf)
```

The arguments in the call to `outfld` are the 8-character field name, the variable array in which it is stored, the first dimension of the data array, the latitude index, and the name of the history buffer. In adding a call to `outfld`, the user need only modify the first three arguments in the list (the last two will always be `lat` and `hbuf`).

Once these steps are taken, the field may either be added to a history file using namelist variables `PRIMARY` or `AUXF`, or removed from the history file via the namelist variable `EXCLUDE`.

Case 2: The variable is new and diagnostic in nature.

The new field should be in a user-defined local array or in the model buffer, if the information needs to be carried across time levels. To add a field to the model buffer requires the following steps:

- Add buffer field pointers in common block `/comgrd/`
- Set those pointers in routine `points`
- Change parameters that determine buffer lengths in `pagrid.h`. These parameters are illustrated in Figure 3.2 (`PBFLNB`, `PBFLNA`, `PLENALCL`, and/or `PCNST`).
- If the variable is used in routines `linemsbc` or `linemsac` or below, pass the variable as an argument into the appropriate routine(s).

Follow directions under Case 1 to include the new variable in a history file.

4.3 Adding New Constituents

Adding tracers to the model is achieved by modifying the parameters `PCNST` and/or `PNATS`. `PCNST` represents the number of advected constituents transported in the atmosphere whereas `PNATS` determines the number of non-advected constituents. CCM3.6 carries one advected constituent (water vapor) by default, but has the capability of to transport up to 98 additional constituents depending upon available memory. It is important to realize that any new constituents will act strictly as passive tracers since there is no feedback mechanism to allow a constituent to affect the predicted climate. However, certain physical processes within the atmosphere do affect the transport of a constituent. Currently, no physical parameterizations act upon non-advected constituents. For example, if the user desires the added non-advected constituents to be vertically diffused, the appropriate sections of the vertical diffusion code will have to be modified. Typical examples of processes that a user may include to influence a new constituent are:

- Define surface fluxes and atmospheric source and sinks of the new constituent.
- Add "adjustment" processes for the constituent.
- Define a minimum value that the constituent at each grid point should always be greater than (default is 1x10-12)

For each constituent added by the user, the model will:

- Create array space and index variables for the new constituents and associated diagnostics.

- Advect all user-advected constituents using SLT.
- Apply vertical diffusion, convective transport and time-filtering to the user-advected constituents.
- Write constituent forecasts and associated diagnostics to the primary history file.
- Check that the constituent values do not fall below an expected minimum (e.g. 0.0).

Constituents are input to and output from the model as dry mass mixing ratio, whereas within routines `aphys`, `tphysbc`, and `tphysac` the units are converted to wet mass mixing ratio.

The model determines index variables for user-added constituents at startup. In particular, user-advected constituents are given the indices `ixuadv` to `ixuadv+nusr_adv-1`, where `nusr_adv` is an input namelist variable. Similarly, user non-advected tracers are referenced by the indices `ixunad` to `ixunad+nusr_nad-1`, where `nusr_adv` is an input namelist variable. All references to these constituents must be made via the appropriate index into the array `q3` in routines `inidat`, `linemsbc` and `linemsac` or into the array `qm1` in routines below `linemsbc` and `linemsac`. The *Common* block for `ixadv` and `ixnad` are contained in the header file `comindx.h`.

To add one or more constituents, the user must make the following modifications.

- Increase the **cpp** token `PCNST` and/or `PNATS` (in `params.h`) by the number of constituents added.
- Determine initial values for the new constituents (see below).
- Modify the code to define surface fluxes and atmospheric sources and sinks and/or "adjustment" processes for the new constituents (see below).

4.3.1 Initializing New Constituent Data Fields

If new constituents are added, the initial dataset (as indicated by namelist variable `NCDATA`) must be altered to include initial constituent data or routine `inidat` must be modified to initialize the constituents. New constituent field data is output to the primary history file by default (see Table 2.13).

If values for the new constituents are added to the initial dataset, the field name of the new constituents must match the field names assigned by the model and initial values must be supplied for every grid point at the given resolution. Constituent field names are assigned in routine `initcom` and saved in array `tracnam(PCNST+PNATS)`. Constituent-related arrays (tendencies, fluxes, etc.) are stored in common block `/comtrcnm/` in header file `comtrcnm.h`. Since water vapor is a transported constituent, its field name (`Q`) occupies the first element of `tracnam`. Thus, the field name of the first new constituent will occupy the second element of `tracnam`, and so on. A simple naming convention is used for all new constituents. It consists of two letters ("`TR`" for advected tracers and "`NA`" for non-advected tracers) followed by a two-digit number. Numbering of the tracers starts at `02` and increments by one for each new constituent.

For example, if two advected and two non-advected constituents have been added, then their assigned field names would be `TR02`, `TR03`, `NA04`, and `NA05`. The field names on the initial dataset must match these names. Once the initial condition dataset has been created, the model will read the new constituent data into the proper array space (no new array space needs to be defined by the user) if the namelist variable `READTRACE` is set to `TRUE`.

Alternatively, the user may opt to initialize the new consituents by adding new code to routine `inidat` and setting namelist variable `READTRACE` to `'.FALSE'`. To modify `inidat`, the user should locate the *if-then-else* block located just beyond the line commented as "`C Initialize tracers if not read in from input data.`" and change the subsequent code accordingly.

4.3.2 Surface Fluxes, Sources, and Sinks

The user may wish to add surface fluxes and atmospheric sources and sinks that influence constituent values. Such code should be added to routine `typhsac` immediately prior to the call to `vdintr`. Surface flux tendencies ($Kg/m^2/sec$) should be stored in array `cflx` beginning at `cflx(1,ixadv)` for the first user-advected tracer, `cflx(1,ixadv+1)` for the second user-advected tracer, `cflx(1,ixnad)` for the first user non-advected tracer, `cflx(1,ixnad+1)` for the second user non-advected tracer, and so on. The first column of this array, `cflx(1,1)`, is reserved for the surface flux of water vapor. Sources and sinks should be computed as tendencies ($Kg/Kg_{air}/sec$) and stored in array `srcsnk`, beginning at `srcsnk(1,1,ixadv)` and/or `srcsnk(1,1,ixnad)`. The user should not add these surface fluxes, sources, and sinks to the constituent fields since the model will do so. The surface flux and source/sink tendency fields, `SFxx` and `SSxx`, are already in the the Master Field List. The user, however, must insert calls to `outfld` within the code (see "Adding New Variables") and use the namelist variables `PRIMARY` and/or `AUXF` to obtain these fields on the history files.

4.3.3 Adjustment Processes

Code to perform adjustment physics (other than convective transport) to the constituents should be inserted within routine `aphys`. There is no pre-existing code to do the adjustment.

4.3.4 Global Minima

During a model run, constituent values may fall below an "expected" minimum. Array `qmin(PCNST)`, set in `initcom` and stored in common block `/comqmin/`, contains the minimum acceptable value for each advected constituent. The first element of this array, `qmin(1)`, holds the minimum value for water vapor. Successive elements hold minimum values for new advected constituents. `qmin` is set to 1x10-12 ($Kg/Kg_{air)}$ for water vapor and to 0.0 for each non-H2O advected constituent (except for one special case in routine `vdiff` where `qmincg(1)=0.0` is used as the water vapor minimum). The model compares the advected constituent values with `qmin` values in routine `qneg3`. Any point whose value falls below the given minimum is reset to the minimum value and a warning message is printed.

NOTE: When this procedure is invoked, global mass is **not** conserved. Also, the default minima may **not** be appropriate for the advected constituents specified by the user. The `qmin` values may be reset by the user in the routine `initcom`. In cases where the model is carrying an advected constituent that is not positive definite (e.g. potential vorticity), `qmin` may be set to a large negative value.

4.4 Changing Resolution

Vertical and/or horizontal resolution can be changed from the model default settings by modifying the appropriate values in `params.h` . "`PLON`" and "`PLAT`" define horizontal resolution in gridpoint space, while "`PTRM`", "`PTRN`", and "`PTRK`" define the horizontal truncation in spectral space. The number of vertical levels is defined by "`PLEV`". Physical parameterizations and/or namelist variables such as the timestep and diffusion coefficients may also have to change with differing resolutions. The off-line codes **interpic**, **definesurf** and **interpsst** can be used to interpolate input datasets to different resolutions (see Sec. 2.3.1.1).

4.5 Adding Parameterizations

New parameterizations should be independent of the order in which model arrays are stored in memory. This implies that if a parameterization requires two time levels of a particular field, these two levels should be passed as input by the interface routine. Parameterizations should also provide separate output arguments for return to the calling routine. Furthermore, they should not modify input arguments which point into the main model buffers or other data structures. Time tendencies may be calculated from the input and output arguments at the level of the parameterization interface routine.

New parameterization variables that must be present at restart must appear on the restart dataset. If a new field is stored in one of the model buffers, it will automatically be written to the restart dataset. Otherwise, code must be added to routine `wrtrs1` to write the new field to a the restart dataset. Code must also be added to routine `splitf` to read the new field upon restart. Since these routines have blocks of code for both in-core and out-of-core cases, changes must be made to both blocks to ensure proper restart capability.

4.6 Trouble-Shooting Model Changes

If the cause of abnormal termination is unclear, ensure that the model is run single-threaded. Abnormal termination in a multitasked job can result in confusing ancillary error messages.

If the model terminates with an internally generated error message, the first place to look is "Appendix C: Model Error Messages" for guidance as to the nature of the abnormal end. However, this error message may be insufficient to determine what must be done to fix the simulation (e.g., just because dry adjustment failed to converge does not explain **why** it happened). Sometimes there is no internally generated error message, as for example when a floating point error occurs. The purpose of this section is to suggest courses of action to determine the nature of the problem when the model terminates abnormally.

Resource allocation errors will be addressed first, followed by remedies for suspected coding errors. Finally, analysis tools are described for physics formulation errors (i.e., where there is an error in modifications to a prognostic variable calculation).

4.6.1 Resource Allocation Errors

SSD storage allocation for scratch files may be specified on the `#QSUB -lQ` line in the batch job script (see "Running the Model"). If insufficient storage is requested, the model will print an error message and terminate. To determine how much SSD storage is required, refer to the line of model printout which says:

```
Total size of files normally assigned to SDS:
```

The user's `#QSUB -lQ` request must at least match this number.

An insufficient memory request (`#QSUB -lM`) will abort the run with a possible error message of the form:

```
Not enough space
```

Check the line of **segldr** output which reads:

```
Program length- 22363070 octal 4843064 decimal
```

Determine that the number of megawords requested on the `#QSUB -lM` line is larger than the program length in decimal. If the initial program length is less than the `#QSUB` request, but only by a few hundred thousand words, a memory allocation problem may still exist since the operating system often needs to allocate additional memory for its own purposes. If sufficient memory is not available, this may not be evident from the error message received. Therefore, if the model is using close to the amount of memory declared on the `#QSUB -lM` line and it appears to be dying due to some system error (e.g.: `PUTFIL:Error in ishell call,ier= 256`), try increasing the memory request by approximately 1 million words and rerun the job.

Another system resource problem occurs on the SUN architectures. The default stack size on the SUN is sometimes too small for larger resolution runs or when running on multiple processors. The model usually fails with a segmentation fault. To set the stack size to its maximum you can use the `limit` command. Typing `limit` alone will print the system resource limits. To set the stack size to its maximum type `limit stacksize unlimited` . Once the stack size has been increased try running the model again. If the stack size was too small before it should run to completion. When running the message passing code on multiple processors it is necessary to place this limit command in your shell startup script. Since the message passing software usually starts new processes to run under, you must make sure that these processes have the larger stack size when started by MPI. An easy way to determine that new shells have the larger stack size is to execute the command `rsh` *`machine`* `limit` (where *machine* is the name of a computer on which to start the remote shell.)

4.6.2 Coding Errors

The following is an ad-hoc list of tips to assist the user who has modified the source code in tracking down coding errors.

The **flint** utility is useful for finding argument list mismatch between routines, improper variable references (i.e. variable is referenced before it is set), and other serious coding errors such as *common* block length mismatches between routines.

Given reasonable output from **flint** and continued suspicion of a coding error, there are some compiler options that may help if run-time debugging becomes necessary. We recommend that the model always be run single-threaded during run-time debugging. It is important to determine early on whether user changes adversely affects only multitasked execution. Also, output generated as a result of these options is much easier to interpret if generated during single-threaded execution. Furthermore, running the code at T5 resolution may speed-up the debugging process.

The **-Rb** command line argument to the Cray f90 compiler enables array bounds checking during model execution.

The **-G1** option to the Cray f90 compiler can also be helpful for run-time debugging. This option produces debug symbol table information.

We suggest that statically allocated memory locations be initialized to " indefinite". (the **-f indef** option to **segldr**). A more costly but equally useful option for debugging is the **-ei** compiler option. This option initializes **stack** memory to indefinite and will likewise result in an error if memory so initialized is used in a floating-point computation. It is much more expensive than static initialization because memory values must be set each time stack memory is made available to a routine. This means a very large amount of memory must be initialized every timestep. Runs using this option will incur a performance penalty of approximately 10%. Compiler option **-ei** is recommended for model development and debugging purposes only.

4.6.3 Formulation Errors

Adding a new physics parameterization to the model or changing the settings for various namelist variables can sometimes result in a model abort. Hopefully, an informative message will be printed to provide a starting point for debugging (see "Appendix C: Model Error Messages"). Where the model dies is not always the same place where things first start going wrong, however.

Some of the tools mentioned in the previous section ("Coding Errors") can also be helpful if a formulation error is suspected. In particular, use of the `-G0` compiler option followed by the **debugview** command is a very useful diagnostic tool if the model is actually crashing.

If the model is running but producing incorrect or suspicious history files, a quick and easy-to-use diagnostic program, **cprtpsf90**, is available. This program provides a statistical analysis of differences in history file data. No command line arguments are required. **cprtpsf90** compares fields of the same name on each file, printing out statistics about the number of differences found, location and magnitude of worst absolute difference, location and magnitude of worst relative difference, rms difference, maximum and minimum field values, and average field values.

References

Acker, T.L., L.E. Buja, J.M. Rosinski, J.E. Truesdale, 1996: *User's Guide to NCAR CCM3*, NCAR Technical Note/TN-421+IA, Boulder, Colorado.

Bath, L.M., J. Rosinski, and J. Olson, 1992: *User's Guide to NCAR CCM2*, NCAR Technical Note/TN-379+IA, Boulder, Colorado.

Berger A. 1977: *Long-Term Variations of the Earth's Orbital Elements*, Celestial Mechanics 15

Bonan, G.B., 1996: *A Land Surface model (LSM version 1.0) for Ecological, Hydrological, and Atmospheric Studies: Technical Description and User's Guide*, NCAR Technical Note/TN-417+STR, Boulder, Colorado.

Bryan, F.O., B. Kauffman, W. Large, and P. Gent, 1996: *The NCAR CSM Flux Coupler,* NCAR Technical Note NCAR/TN-424+STR, Boulder, Colorado.

Buja, L., 1992: *Introduction to the UNICOS CCM Processor*, NCAR Technical Note NCAR/TN-383+IA, Boulder, Colorado.

Collins, W.P.,1998:A Global Signature of Enhanced Shortwave Absorption by Clouds, JGR, submitted.

Hack J.J., B.A. Boville, B.P. Briegleb, J.T. Kiehl, P.J. Rasch, D.L. Williamson, 1993: Description of the NCAR Community Climate Model (CCM2)

Kalnay, E., M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. Stackpole, J. Tucillo, L. Umscheid, and D. Williamson, 1989: Rules for Interchange of Physical Parameterizations. *Bulletin of the American Meteorological Society*, 70, 6, 620-622.

Kiehl, J.T., J. Hack, G. Bonan, B. Boville, B. Briegleb, D. Williamson, and P. Rasch, 1996: *Description of the NCAR Community Climate Model (CCM3)*, NCAR Technical Note NCAR/TN-420+STR, Boulder, Colorado.

Williamson, D.L. and P.J. Rasch, 1989: *Two-Dimensional Semi-Lagrangian Transport with Shape Preserving Interpolation*, Mon. Wea. Rev., 117, 102-129.

Appendix A: Glossary

Name	Synopsis
branch run	A type of continuation run of the CCM. A branch run starts a new case using the restart files from a previous model run.
case	A term used to denote a model experiment, including one initial run and as many continuation runs as required to conclude the experiment. A model case is cataloged using the `CASEID` namelist variable, which appears in word 1 of the character (second) history file header record.
CCM	Community Climate Model; one of a series of codes produced at NCAR for use in making climate predictions.
CCM Processor	A program available from the CCM Core Group which post-processes CCM history files on the NCAR computing system. Currently it is available, but not supported (Buja, 1992).
ccm2nc	CCM to netCDF history file converter program. Reads in a CCM history file and outputs an equivalent netCDF file.
CSM	Climate System Model.
cpp	C preprocessor.
continuation run	A type of run that uses a restart file from a previous run to initialize the data fields (no initial dataset is read). Restart and branch runs are each possible types of a continuation run.
distributed memory	Multi-processing on multiple CPU's using the SPMD programming model.
DOM	Data Ocean Model. Ocean component of stand-alone CCM that reads in SST information from an input dataset.
ECMWF	European Center for Medium Range Weather Forecasts

FFT	Fast Fourier Transform.
gzip	GNU decompression utility similar to UNIX uncompress. Takes a file and inflates it to it's full size so that it can be used. Files are compressed to save disk space and save on network file transfer times.
header file	A file containing common blocks and variable declarations and named with a `".h"` suffix (e.g. `comhst.h`).
heap	Memory that is dynamically allocated by the system. Unlike stack memory, heap memory can be allocated or de-allocated at any point during program execution.
history file	A binary dataset that is the primary output for field values generated by the model during the course of a time integration.
LSM	Land Surface Model
MSS	The NCAR Mass Store System.
multitasked	A program configured to execute on more than one central processor simultaneously.
netCDF	Network Common Data Format. Self describing, platform independent binary data format (created by UNIDATA).
out-of-core	Refers to the use of secondary storage instead of main memory.
packing	A process optionally applied to output history file data, which compresses the values according to a specified density. Data can be packed either 2, or 4 to 1.
pcnst	Number of advected constituents carried in the model.
plat	Number of Gaussian latitudes on the transform grid.
plev	Number of vertical model levels.
plon	Number of longitudes on the transform grid
MPI	Message Passing Interface. A standardized library for distributed memory parallel processing.

restart run	A type of continuation run of the CCM. A restart run continues a previous run from its point of termination, by reading most recent restart
single-threaded	Parallel-processing term. Refers to code that executes on only one processor.
SDS	Secondary Data Segment, a file management method available via UNICOS software for use on the SSD.
shared-memory	Parallel-processing term. Refers to using a machine where multiple CPU's share the same memory.
SLT	Semi-Lagrangian Transport.
SOM	Slab Ocean Model.
SPMD	Single Program Multiple Data.
SSD	Solid-State Storage Device, a very high-performance secondary storage device.
stack	Memory local to a subroutine or function.
transform grid	A grid used to evaluate all nonlinear and diabatic forcing terms in physical space.

Appendix B: CCM3.6 Calling Tree

The following figure depicts the CCM3.6 calling tree when run in shared memory mode (i.e. not SPMD), using Eulerian dynamics. The tree was generated with the tool "flint". Many utility routines were excised from the output for clarity.

Figure B. Flowchart.

```
CCM3
+-DATETIME (1)
+-PRESET
+-DATA-+-LINEBUF_STDOUT
|      +-INITINDX
|      +-ORB_PRINT
+-LUNITS-+-NAVU (4)
+-POINTS
+-TRUNC
+-INITAL-+-GETFIL (5)-+-SHELL_CMD (6)-+-(ISHELL)
|        +-READINITIAL-+-GETFIL see 5
|        +-INITCOM (8)-+-SMALLEST
|        |             +-HYCOEF
|        |             +-GAUAW (9)-+-BSSLZR
|        |             +-SETTAU (10)-+-QREIG-+-CMPHES
|        |             |             |       +-CMPLR
|        |             |             +-(MINV)
|        |             +-PHCS
|        |             +-REORDP
|        |             +-(SET99)
|        +-CALENDR (12)
|        +-INIDAT-+-(RANSET)
|        |        +-CELL_BOX (13)-+-GAUAW see 9
|        |        +-BILIN
|        |        +-VERTINT
|        |        +-SGHPHIS-+-GETFIL
|        |        |         +-GETOPO
|        |        |         +-CELL_BOX see 13
|        |        |         +-BINF2C
|        |        |         +-VARF2C
|        |        |         +-MAX_OVR (18)--CELL_EDGE (19)-+-GAUAW see 9
|        |        |         +-MAP_I (20)-+-CELL_EDGE see 19
|        |        |         |            +-CELL_AREA
|        |        |         |            +-AO_I-+-AO
|        |        |         |            +-AREA_AVE
|        |        |         +-SM121
|        |        +-(RANF)
|        |        +-(FFT991)
|        |        +-SPETRU
```

```
|        |        +-CHEMMIX
|        |        +-INITESTTR
|        |        +-QNEG3 (21)
|        |        +-WRITERIC (22)--IOSTOP (23)-+-(ISHELL)
|        |        +-PDELB0
+-RESUME-+-RESTRT-+-GETFIL see 5
|        |        +-READRG-+-IOSTOP see 23
|        |        +-SPLITF-+-READARR (24)--IOSTOP see 23
|        |        |        +-WRITERIC see 22
|        |        +-READARR see 24
|        |        +-WRITERIC see 22
|        |        +-RDHDR
|        |        +-BLDFLN (25)-+-CALENDR see 12
|        +-INITCOM see 8
|        +-CALENDR see 12
+-INTI-+-MGWINTI
|      +-VDINTI--PBINTI
|      +-TSINTI
|      +-RADINI
|      +-CAINTI
|      +-ESINTI--GESTBL-+-GFFGCH--ENDRUN see 3
|      +-MFINTI
|      +-HDINTI
+-INITEXT-+-GETFIL see 5
|         +-OZNINI-+-BNDDYI (26)
|         |        +-LININTERP (27)
|         |        +-BILIN see 14
|         +-CHEMINI-+-GETFIL see 5
|         |          +-BNDDYI see 26
|         +-ORB_PARAMS--{ABORT}
|         +-LSMINI-+-LSMCTLI-+-SHELL_CMD see 6
|         |        +-MSSPNI
|         |        +-GETFIL see 5
|         |        +-CELL_BOX see 13
|         |        +-FSURDAT-+-CELL_BOX see 13
|         |        |         +-MAX_OVR see 18
|         |        |         +-LANDTYP-+-GETFIL see 5
|         |        |         |         +-MAP_I see 20
|         |        |         |         +-RANK
|         |        |         +-SOILCOL-+-GETFIL see 5
|         |        |         |         +-MAP_I see 20
|         |        |         |         +-RANK
|         |        |         +-SOILTEX-+-GETFIL see 5
|         |        |         |         +-MAP_I see 20
|         |        |         |         +-SOILTYP
|         |        |         +-LANDWAT-+-GETFIL see 5
|         |        |         |         +-MAP_I see 20
|         |        |         +-BASINMAT-+-MAP_I see 20
|         |        |         |          +-RANKN
|         |        |         +-PUTFIL (30)-+-(MSWRITE)
|         |        |                       +-{UNLINK}
|         |        +-ALLOCDYN
|         |        +-LSMTCI
|         |        +-BASININI
|         |        +-LSMRES
```

```
|         |        +-HISTTC-+-HISTSLF
|         |        |        +-HISTMLF
|         |        |        +-HISTCRT (31)-+-DATETIME see 1
|         |        |        +-HISTWRT (32)-+-SLF2XY
|         |        |        |              +-MLF2XY
|         |        |        +-PUTFIL see 30
|         |        +-HISTINI-+-HISTLST
|         |        |         +-GETFIL see 5
|         |        +-CALENDR see 12
|         |        +-LSMTVI-+-PUTFIL see 30
|         |        |        +-ECODYN (33)--PHENOL
|         |        |        +-SURALB (34)-+-LSMZEN--ORB_DECL
|         |        |                      +-SNOALB
|         |        |                      +-SOIALB
|         |        |                      +-TWOSTR
|         |        +-ATMINI
|         +-SSTINI-+-BNDDYI see 26
|         +-SSTINT (35)-+-BNDDYI see 26
|         +-SSTAN (36)
|         +-ZENITH (37)--ORB_DECL
|         +-ALBOCEAN
+-INTHT-+-BLDFLD-+-ADDFLD
|       +-BLDFLN see 25
|       +-FLDLST-+-INIFLD
|       +-INTHED-+-DATETIME see 1
|       +-WRITERIC see 22
|       +-GETFIL see 5
|       +-SPLITG-+-READARR see 24
|                +-WRITERIC see 22
+-RGNHBF
+-STEPON-+-GRDINI-+-GRDXY-+-GAUAW see 9
|        |        +-BASDY--LCDBAS
|        |        +-BASDZ--LCDBAS
|        |        +-BASIY--LCBAS
|        |        +-VRTMAP
|        +-SCAN1BC-+-BLDFLN see 25
|        |         +-PREALC (38)-+-(ISHELL)
|        |         +-WRTHDR-+-CALENDR see 12
|        |         |        +-DATETIME see 1
|        |         |        +-PRNTHD
|        |         +-RGNFLS-+-CALENDR see 12
|        |         |        +-PREALC see 38
|        |         +-WRTRS1-+-WRTARR--IOSTOP see 23
|        |         |        +-READRIC (39)--IOSTOP see 23
|        |         +-ADVNCE-+-SETTAU see 10
|        |         |        +-CALENDR see 12
|        |         |        +-OZNINT-+-BNDDYI see 26
|        |         |        |        +-LININTERP see 27
|        |         |        |        +-BILIN see 14
|        |         |        +-SSTINT see 35
|        |         |        +-CHEMINT-+-BNDDYI see 26
|        |         +-LINEMSBC-+-READRIC see 39
|        |         |          +-PLEVS0
|        |         |          +-QNEG3 see 21
|        |         |          +-OMCALC
```

```
|        |         |          +-MR2MF
|        |         |          +-APHYS-+-VIRTEM
|        |         |          |       +-ZMID
|        |         |          |       +-ZINT
|        |         |          |       +-DADADJ--BLOWUP (41)-+-PUTFIL see 30
|        |         |          |       |                     +-{ABORT}
|        |         |          |       +-QNEG3 see 21
|        |         |          |       +-CONV_CCM-+-BUOYAN
|        |         |          |       |          +-CLDPRP
|        |         |          |       |          +-CLOSURE
|        |         |          |       |          +-Q1Q2
|        |         |          |       |          +-CONVTRAN
|        |         |          |       +-CMFMCA-+-AQSATD
|        |         |          |       |        +-VQSATD
|        |         |          |       +-COND--AQSAT
|        |         |          +-MF2MR
|        |         |          +-CPSLHY
|        |         |          +-AQSAT
|        |         |          +-WRITUP--WSHIST-+-(PACKAF)
|        |         |          |                +-WRTHARR
|        |         |          +-VIRTEM
|        |         |          +-POTTEM
|        |         |          +-ZMID
|        |         |          +-TPHYSBC-+-ZENITH see 37
|        |         |          |         +-CLDINT-+-CLDFRC--AQSAT
|        |         |          |         |        +-ZINT
|        |         |          |         |        +-CLDCLW
|        |         |          |         |        +-CLDEFR
|        |         |          |         |        +-CLDEMS
|        |         |          |         |        +-CLDSAV
|        |         |          |         +-RADCTL-+-TORGRID
|        |         |          |         |        +-RADOZN
|        |         |          |         |        +-RADINP--ORB_DECL
|        |         |          |         |        +-AERMIX
|        |         |          |         |        +-RADCSW-+-RADDED
|        |         |          |         |        |        +-RADCLR
|        |         |          |         |        +-RADCLW-+-RADTPL
|        |         |          |         |        |        +-RADOZ2
|        |         |          |         |        |        +-TRCPTH
|        |         |          |         |        |        +-RADEMS-+-TRCPLK
|        |         |          |         |        |        |        +-TRCEMS
|        |         |          |         |        |        +-RADABS-+-TRCAB
|        |         |          |         |        |        |        +-TRCABN
|        |         |          |         |        |        +-WRITERIC see 22
|        |         |          |         |        |        +-READRIC see 39
|        |         |          |         |        +-TRCMIX
|        |         |          |         |        +-FMRGRID
|        |         |          |         +-SRFXFER
|        |         |          +-WRITERIC see 22
|        +-LSMMAIN--LSMDRV-+-HISTEND--CALENDR see 12
|        |                 +-CALENDR see 12
|        |                 +-LSM-+-SURPHY-+-SURRAD
|        |                 |     |        +-SURTEM--STOMATA
|        |                 |     |        +-SOITEM--TRIDIA
|        |                 |     |        +-LAKTEM--TRIDIA
```

```
|        |                 |     +-VERH2O-+-CANH2O
|        |                 |     |        +-SNOH2O
|        |                 |     |        +-INFIL
|        |                 |     |        +-SOIH2O--TRIDIA
|        |                 |     +-SURBGC
|        |                 |     +-ECODYN see 33
|        |                 |     +-SURALB see 34
|        |                 |     +-HISTSLF
|        |                 |     +-HISTMLF
|        |                 +-BASINDRV
|        |                 +-HISTHAN-+-HISTNAM
|        |                           +-HISTCRT see 31
|        |                           +-HISTWRT see 32
|        |                           +-PUTFIL see 30
|        |                           +-RESTWRT-+-PUTFIL see 30
|        +-CCMOCE-+-CALENDR see 12
|        |        +-SSTAN see 36
|        |        +-SRFOCE--FLXOCE
|        |        +-SRFSICE-+-FLXSICE
|        |        |         +-SRFTSB-+-MTDLSS
|        |        +-ZENITH see 37
|        |        +-ALBOCEAN
|        +-SCAN1AC--LINEMSAC-+-READRIC see 39
|        |                   +-PLEVS0
|        |                   +-MR2MF
|        |                   +-RESETR
|        |                   +-TPHYSAC-+-RESETR
|        |                   |         +-QNEG4
|        |                   |         +-RNSFWCRP
|        |                   |         +-RNDECAY
|        |                   |         +-CHEM
|        |                   |         +-VDINTR-+-VDIFF-+-VIRTEM
|        |                   |         |        |       +-PBLDIF
|        |                   |         |        |       +-MVDIFF
|        |                   |         |        |       +-QVDIFF
|        |                   |         |        |       +-QNEG3 see 21
|        |                   |         +-MGWINTR-+-MGWDRAG
|        |                   +-MF2MR
|        |                   +-GRMULT--VIRTEM
|        |                   +-VIRTEM
|        |                   +-ZMID
|        |                   +-(FFT991)
|        |                   +-STATS
|        |                   +-WRITERIC see 22
|        +-WRAPUP-+-BLDCOM
|        |        +-PUTFIL see 30
|        |        +-WSDS
|        |        +-BLDFLN see 25
|        +-SCANSLT-+-SLTINI-+-EXTYV
|        |         |        +-EXTYS
|        |         |        +-EXTX
|        |         |        +-CUBXDR
|        |         |        +-LIMDX-+-SCM0
|        |         +-PLEVS0
|        |         +-QMASSA
```

```
|        |         +-SLTB1-+-SPHDEP-+-S2GPHI
|        |                 |        +-BANDIJ
|        |                 |        +-HRVINT-+-HERXIN
|        |                 |        |        +-LAGYIN
|        |                 |        +-S2GVEL
|        |                 |        +-TRAJMP
|        |                 |        +-G2SPOS
|        |                 |        +-TRJMPS
|        |                 |        +-TRJGL
|        |                 +-HRINTP-+-HERXIN
|        |                 |        +-CUBYDR
|        |                 |        +-LIMDY--SCM0
|        |                 |        +-HERYIN
|        |                 +-CUBZDR
|        |                 +-VRTDEP-+-KDPFND
|        |                 |        +-HERZIN
|        |                 |        +-VDPLIM
|        |                 +-LIMDZ--SCM0
|        |                 +-HERZIN
|        +-DYNDRV-+-DYN
|        |        +-QUAD
|        |        +-TSTEP
|        |        +-COURLIM
|        |        +-HORDIF
|        +-SCAN2--SPEGRD-+-RESETR
|        |               +-GRCALC
|        |               +-(FFT991)
|        |               +-PLEVS0
|        |               +-PDELB0
|        |               +-DIFCOR
|        |               +-QMASSD
|        |               +-XQMASS
|        |               +-WRITERIC see 22
|        +-WHENWR--CALENDR see 12
```

Appendix C: Model Error Messages

All errors trapped by CCM3.6 result in a printed message starting with the name of the subroutine of origin. The following is an alphabetical list of these messages, along with an explanation for each. All errors are fatal unless otherwise stated. Where possible, a suggested course of action is included.

AO error: n_ovr= *mm* exceeded mx_ovr = *nn*,
for output lon,lat = *ii jj*

Dataset inconsistency. Should never occur if the supported datasets are used.

AO_I error: area not conserved for lon,lat = *ii jj*
sum of overlap weights = *nn*

Dataset inconsistency. Should never occur if the supported datasets are used.

BASININI fatal error: discrepancy in land points
expecting *mm* points but found *nn*

Dataset inconsistency. Should never occur if the supported datasets are used.

BILIN: Failed to find interp values

The bilinear interpolation routine was unable to find input array values between which it could interpolate. This is an internal error which should never occur. If it does, a message to ccm-bugs@ucar.edu may be in order.

BILIN: Input x-grid must be between 0 and 360

Input grid for longitude interpolation has coordinate values outside 0-360 degree range. This is required because longitudinal periodicity is assumed.

BILIN: Insane dxinwrap value = *mm* avg=*nn*

Input grid is assumed to be periodic in the longitudinal direction. This error results when the delta-x computed between the last longitude and the first longitude is vastly different than the average intermediate values. Solution: define longitude input and output grids such that the delta-x is periodic.

BILIN: Must have at least 2 input points for
interpolation

Input array to be bi-linearly interpolated must have at least 2 points in longitude, and 2 points in latitude.

BILIN: Non-monotonic coordinate array(s) found

Coordinate arrays for interpolation must increase monotonically. Check input arrays.

BILIN: No overlap in y-coordinates

Error in the latitudinal coordinate of the input arrays to routine bilin. These coordinates must have some overlap for interpolation. Check input and output array coordinates.

BILIN: No overlap in x-coordinates

Error in the longitudinal coordinate of the input arrays to routine bilin. These coordinates must overlap for interpolation. Check input and output array coordinates.

BILIN: *nn* of N/S output grid will have to be extrapolated

The interpolated-to array is larger than the interpolated-from array. Some percentage of the output array will be extrapolated. This is a non-fatal error.

BILIN: Output x-grid must be between 0 and 360

Output grid for longitudinal interpolation has coordinate values outside of 0-360 degree range. The periodicity assumption is why the 0-360 range is a requirement.

BILIN: something bad in indices or weights

This is an internal error which should never occur.

BLDFLD: Too many fields -- pflds,if=*nn mm*

Only `pflds` words of space are allocated for arrays carrying field information. If you have added fields, you must increase `pflds` accordingly in `pagrid.h`.

BLDFLN: maximum output file number exceeded

Largest number of possible files is 9999

BNDDYI: Bad month index = *mm*

The input month number *mm* (from a boundary dataset) was not between 1 and 12.

BNDDYI: bad day of year = *dd*

The input day number dd (from a boundary dataset) was not between 1 and 365.

CALENDR: Invalid base day of base date input: *dd*

The input day number (from the initial dataset or from `NNBDAT` in the namelist) was not valid for the input month.

CALENDR: Invalid base month input: *mm*

The input month number (from the initial dataset or from `NNBDAT` in the namelist) was not between 1 and 12.

`CALENDR: Invalid base seconds(mbsec): `*`ss`*

The input number of seconds (from the initial dataset or from `NNSBAS` or `NNBSEC` in the namelist) was not valid.

`CCM3: ieee trapping not supported here`

The code is compiled for an architecture that does not support IEEE error handling. If you are *not* running on a SUN machine make sure that #define SUN is not set in the params.h file

`CCM3: cpp token CRAY is defined but PVP is not.`
`PVP must be defined for optimal performance on this`
`architecture. If you really want to run this way,`
`delete the stop 999 statement in ccm3.F`

Self explanatory

`CELL_BOX error: lat grid type `*`mm`*` not supported`

Internal algorithmic error which should never occur.

`CELL_BOX error: lon grid type `*`nn`*` not supported`

Internal algorithmic error which should never occur.

`CELL_EDGE error: lat grid type `*`mm`*` not supported`

Internal algorithmic error which should never occur.

`CELL_EDGE error: lon grid type `*`nn`*` not supported`

Internal algorithmic error which should never occur.

`CHEMINI: Error in horizontal interpolation`

Error occurred when trying to horizontally interpolate the input tracer data to the model latitudes.

`CHEMINI: Error in vertical interpolation`

Error occurred when trying to vertically interpolate the input tracer data to the model levels.

`CHEMINI: error when cycling data:1st month must be 1`

Error in input tracer data: first month must be 1.

`CHEMINI: error when cycling data:last month must be 12`

Error in input tracer data: last month must be 12.

CHEMINI: Failed to find dates bracketing mcdate, mcsec= *yymmddss*

The model integration time does not fall between available dates on the input tracer dataset.

CHEMINI: Failed to open file *filename*

I/O error while opening a file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

CHEMINI: latsiz=*ll* must = ptrlat=*nn*

Tracer dataset does not have number of latitudes specified in header file `comchem.h`. Modify `comchem.h` accordingly.

CHEMINI: levsiz= *ll* must = ptrlev=*nn*

Tracer dataset does not have number of levels specified in header file `comchem.h`. Modify `comchem.h` accordingly.

CHEMINI: longitude dependence not implemented

Tracer dataset is currently assumed to have no longitudinal dependency

CHEMINI: lonsiz= *xx* must = ptrlon=*yy*

Tracer dataset does not have number of longitudes specified in header file `comchem.h`. Modify `comchem.h` accordingly.

CHEMINI: timsiz=*xx* must = ptrtim=*yy*

Tracer dataset does not have number of time samples specified in header file `comchem.h`. Modify `comchem.h` accordingly.

CHEMINT: Attempt to access bad month

The model integration time does not fall between available dates on the input tracer dataset.

CHEMINT: Bad fact1 and/or fact2=*xx yy*

Error interpolating to a daily value. The interpolation factors do not agree to with in 32 bit round-off.

CHEMINT: Input data for date *yymmdd*
does not exceed model date *yymmdd* sec *ss* Stopping.

Model time does not coincide with dates for chemical loss rate data. Check namelist date variable

CMPLR: Entered with incorrect dimension NC=*nn*

Subroutine `CMPLR` was entered with a dimension for the upper Hessenberg matrix which is less than or equal to zero.

CMPLR: Failed to converge in *mm* iterations Eigenvalue=*nn*

The algorithm that computes the Gaussian weights has failed to converge in *mm* iterations. The user can increase the number of iterations that the algorithm performs by increasing the value of the local variable `nfail`.

COURLIM: Courant limit exceeded at k,j= *kk jj* (estimate = eee.eee), solution has been truncated to wavenumber *nn* ***

The CFL limit has been exceeded. Therefore the high-frequency end of the spectrum has been truncated at the level specified by namelist variable `KMXHDC`. Not a fatal error.

CUBYDR: Departure point out of bounds: jcen, icount,pgls= *ll mm nn*

This is an indication that the model is in the process of blowing up. Check that namelist variable `DTIME` is appropriate for the resolution.

DADADJ: No convergence in dry adiabatic adjustment

Model has entered a non-physical state. User will have to analyze state variables at the latitude and longitude given in the error message.

DADADJ: Convergence criterion doubled to EPS= nn.nnnn for DRY CONVECTIVE ADJUSTMENT at Lat,Lon= *ii jj*

Nonfatal error - dry convective adjustment algorithm has failed to converge within the default convergence limit given the current state of the model. The convergence criterion will be doubled and the algorithm will repeat the convergence tests.

DATA: Namelist variable CASEID must be set

The case-id must be defined via the namelist parameter `CASEID`.

DATA: Cannot find LOGNAME environment variable

This error would indicate a system problem, since `$LOGNAME` is a system environment variable.

DATA: Cannot find HOME environment variable

$$$$$This error would indicate a system problem, since `$HOMEDIR` is a system environment variable.

DATA: cldw_adv and cldw_nad cannot BOTH be true

Prognostic cloud water (not available in CCM3.6) cannot be both advected and not advected.

DATA: For BRANCH run, NREVSN must be a full pathname.

Because the case name changes on a branch run, the user must specify the full pathname for

NREVSN.

DATA: inithist must be set to monthly or yearly

The model namelist variable for generating an initial condition file can only assume the values of 'MONTHLY' or 'YEARLY'

DATA: Invalid averaging flag input for tape t Flag = F

`NINAVG` must be one of the following: `A` averaged `I` instantaneous `M` minimum `X` maximum `NINAVG(1)` may also be equal to `Q` for a monthly average

DATA: ERROR
Invalid averaging flag input for tape t
Flag = F
Monthly average only allowed for history file 1

Self explanatory.

DATA: Error, model not designed to run multi-constituents while running the idealized dynamical core of the model. For "idealdyn = .true.", set pcnst=1 and pnats=0.

If namelist variable `IDEALDYN` is set to true, be sure that `pcnst` and `pnats` (found in header file `params.h`) are set to 1 and 0, respectively.

DATA: Invalid packing density *n*

Packing density `NDENS` must be between 1 and 4.

DATA: iradae must be an even multiple of iradlw. iradae = *ii* iradlw = *jj*

The absorptivity/emissivity calculation must fall on a long-wave radiation timestep.

DATA:KMXHDC must be between 0 and plev-1

`KMXHDC`, an input variable, is the number of levels over which to apply the Courant limiter.

DATA: Must input either nestep or nelapse

The user must specify an ending time for the model run.

DATA: Must specify non-zero nhtfrq

The history file write frequency for one of the declared history files is zero.

DATA: Must specify only one topographic file
TOPOFILE and TOPOFILENC namelist vars are both defined

Use either TOPOFILE *or* TOPOFILENC but not both

DATA:*NOTE: Extra overhead invoked putting a/e numbers on restart dataset*** To avoid, make mod(nhtfrq,iradae) = 0**

Not an error. There will be a restart file generated for absorptivity/emissivity values since the history file write does not necessarily occur on an absorptivity/emissivity timestep, as defined by `IRADAE`.

`DATA: Starting file number (stfnum) too large. Max = 9999. Stfnum = `*`ssss`*

Output file numbering starts at `STFNUM`, which must not be larger than the largest allowable number (4 digits).

`DATA: The namelist variables INCORHST,INCORRAD,INCORBUF must be .TRUE. for a SPMD run.`

The user has declared `INCORHST`, `INCORRAD`, and/or `INCORBUF` to be false in the namelist. All three namelist variables must be true for an SPMD run.

`DATA: The regeneration option is no longer available`

CCM3 no longer supports regeneration of history files. History files can be recreated by making a branch run from a restart file. Valid values for namelist variable `NSREST` are 0, 1, or 3.

`DATA: the value of NREFRQ must be 1 or 0`
`Resetting NREFRQ to 1`

The namelist variable `NREFRQ` determines whether restart files will be written. It can assume a value of 0 (no restart files) or 1 (restart files written.)

`DATA: Too many auxiliary files declared.`

The user has declared more than `ptapes` auxiliary history files via the input *variable* `AUXF`. Users who want more than 5 auxiliary files must increase `ptapes` in `pagrid.com`.

`DATA: ************* Warning *********** Namelist variable NREFRQ is invalid for averaging. Use MRESFQ to adjust restart`

The user must use namelist variable `MRESFQ` to request mid-month restarts on a monthly average run.

`DECOMP: Latitude decomposition failed for NPES=`*`nn`*`, lat=`*`ll`*

Check to make sure the number of latitudes is greater than or equal to the number of processors. Only applies to message passing version of CCM3

DECOMP: npes(*mm*) must be <= plat(*nn*)

When CCM3.6 is configured to run in distributed-memory (SPMD) mode, the maximum number of processes is the same as the number of Gaussian latitudes. Solution: set environment variable `NPES` to be no higher than the number of Gaussian latitudes.

DECOMP: Number of procs(*nn*) must be a multiple of 2

When CCM3.6 is configured to run in SPMD mode, the number of processes (defined by environment variable `NPES`) must be even.

DECOMP: Ran out of work to parcel to processors

An internal algorithmic problem has occurred in the message-passing implementation. Contact ccm-bugs@ucar.edu

DECOMP: Something wrong in computation of northern/southern neighbors

An internal algorithmic problem has occurred in the message-passing implementation. Contact ccm-bugs@ucar.edu

DECOMP: Workleft not zero. Value is *ii*

An internal algorithmic problem has occurred in the message-passing implementation. Contact ccm-bugs@ucar.edu

DRAINBAS: nlon_i and nlon_ncd do not match
nlon_i = *mm* nlon_ncd = *nn*

Dataset inconsistency. Should never occur if the supported datasets are used.

`DRAINBAS: nlat_i and nlat do not match`
`nlat_i = `*`mm`*` nlat_ncd = `*`nn`*

Dataset inconsistency. Should never occur if the supported datasets are used.

`DRAINBAS: total weight of zero encountered`
`(DRAINBAS): io,jo = `*`ii jj`*

Dataset inconsistency. Should never occur if the supported datasets are used.

`DRAINBAS: normalization incorrect`
`(DRAINBAS): io,jo = `*`ii jj`*

Dataset inconsistency. Should never occur if the supported datasets are used.

`FACTOR: `*`nn`*` has a prime factor other than 2, 3,or 5. Aborting...`

Only applies to SPMD. Due to FFT constraints the number of latitudes must have no prime factors other than 2,3, or 5.

`FACTOR: Problem size is not divisible by 2`

When running the message passing version of CCM3, the number of latitudes must be an even number.

`FLDLST: cant find auxiliary field `*`fieldname`*

Input namelist variable `AUXF` requested a field not in the Master Field List.

FLDLST: exceed number of auxil. fields *ii*

Too many auxiliary fields specified in namelist variable `AUXF.` Increase Fortran parameter `pflds`

FLDLST: exceed number of history fields *ii*

Too many history fields specified. Increase Fortran parameter `pflds`.

`FLDLST: Too many fields for all history files (primary and auxillary) -- pflds,kfld=`*`ii`*`,`*`jj`*

Too many history fields specified. Increase Fortran parameter `pflds`.

`FREEMEM: Bad return code = `*`errcode`*` from hpdeallc: Abort [Message]`

Cray library routine **hpdeallc** encountered an error while attempting to return dynamically-allocated memory.

`GAUAW: Error exit, no convergence in 10 iterations`

The iterative technique to find abscissas for the Gaussian integration failed to converge in 10 tries. Check that epsilon (`gauaw` local variable `eps`) is consistent with machine precision.

`GESTBL:FATAL ERROR********************************* TMAX AND TMIN REQUIRE A LARGER DIMENSION ON THE LENGTH OF THE SATURATION VAPOR PRESSURE TABLE ESTBL(PLENEST) TMAX, TMIN, AND PLENEST = ffff.ff gggg.gg iii`

The `gestbl` procedure requires that the parameter `plenest` (in `eslookup.h`) be `.ge. tmax - tmin + 2` (where `tmin` and `tmax` are set in `esinti`). Reset *parameter* `plenest` to satisfy this condition.

`GETAVU: Ran out of fortran unit numbers`

CCM3 is only capable of using 99 unit numbers for open files. If the user has added additional calls to `navu` or `getavu` then they must make sure all units are freed (call to `relunit`) when those same Fortran files are closed.

`(GETFIL): failed cmd= `*`commandstring`*

A shell command failed. The command itself is printed out

`(GETFIL): local filename has zero length`

An input pathname must include an actual filename.

`GETMEM: Bad return code = `*`errcode`*` from hpalloc: Abort [Message]`

Cray routine **hpalloc** encountered an error while attempting to dynamically allocate memory. Solution: increase memory limit (`#QSUB -lm` if running in batch mode on a Cray PVP machine).

`GETUNIT: Unit `*`uu`*` is in use. Pick another one`

User is trying to assign a unit which is already in use. Make sure the unit is free.

GFFGCH:FATAL ERROR **************************** TRANSITION RANGE FOR WATER TO ICE SATURATION VAPOR PRESSURE, TR, EXCEEDS MAXIMUM ALLOWABLE VALUE OF 40.0 DEGREES C TR = ffff.ff**

This routine assumes that the maximum size of the transition region for calculating saturation vapor pressure over water and saturation vapor pressure over ice is 40 degrees C. The variable `trice` in `esinti` must be set to be `.le. 40` to avoid this error.

GETUNIT: Unit *uu* is in use. Pick another one

Requested Fortran unit number has already been assigned.

HERXIN: Fatal error: ppdy must be set to 4

parameter `ppdy` in `parslt.h` must be set to 4 for the semi-Lagrangian code to run properly.

(HISTCRTINI): number initial history fields maximum allowed

Number of LSM initial dataset fields is greater than LSM parameter `malflds` set in `lsmpar.h`

(HISTCRTTC): number time constant history fields maximum allowed

Number of LSM time constant dataset fields is greater than LSM parameter `malflds` set in `lsmpar.h`

histlst error: attempting to change field type for field = *field* to inaccurate type = nn. Valid types are: *aa,bb,cc,dd*

Attempting to use non-supported field type for output field. Modify namelist input.

histlst error: number fields maximum allowed

Number of total output fields greater than parameter `malflds` in header file `lsmpar.h`. Increase `malflds`.

histlst error: number single-level fields maximum allowed

Number of single-level -output fields greater than parameter `mslflds` in header file `lsmpar.h`. Increase `mslflds`.

histlst error: number multi-level fields maximum allowed

Number of multi-level -output fields greater than parameter `mmlflds` in header file `lsmpar.h`. Increase `mmlflds`.

histnam error: nyr = *yyyy* is 9999

Year number in LSM history file name (for monthly averaged files) is greater than 9999.

histnam error: nfil = *filenumber* is 99999

File number in LSM history file name (for monthly averaged files) is greater than 9999.

histtc error: number single-level fields maximum allowed

Number of single-level -output fields greater than parameter `mslflds` in header file `lsmpar.h`. Increase `mslflds`.

histtc error: number multi-level fields maximum allowed

Number of multi-level -output fields greater than parameter `mmlflds` in header file `lsmpar.h`. Increase `mmlflds`.

(HISTWRT): could not match single level 1d flds
(HISTWRT): 1d field name : *filename*

In trying to write out 1d fields, requested field name does not match any output fields. Modify LSM namelist accordingly.

(HISTWRT): could not match multi level 1d flds

In trying to write out 1d fields, requested field name does not match any output fields. Modify LSM namelist accordingly.

HYCOEF: A and/or B vertical level coefficients at full levels are not the arithmetic mean of half-level values

This is a non-fatal error.

INIDAT: Attribute length of from_hires is too long

Attribute length of initial dataset netCDF variable `from_hires` cannot be creater than 256

INIDAT: Bad number of dims or ordering on 2-d fld

Dimensions of initial dataset fields must be ordered `lon,lev,lat` for multi level fields and `lon,lat` for single level fields

INIDAT: ERROR - Dimensions incorrectly specified on initial dataset. Expecting dimension names of time,lat,lon, and lev (case sensitive)

Self explanatory.

INIDAT: Initial dataset dims incompatible with model

Number of longitudes, latitudes and levels on initial dataset conincide with model settings

INIDAT: Local space not large enough for initial dataset record.Local parameter phtbf must be at least *nn*n

The initial dataset record size is too large for the locally declared buffer in `inidat`. Increase the size of parameter `phtbf`, local to `inidat`, that defines the size of space allocated for reading a data record from the initial dataset.

INIDAT: Local space not large enough for initial dataset record
Local parameter phtbf must be at least *nn*

Increase the size of `phtbf` to be at least `nn`.

INITAL: Failed to open file *filename*

I/O error while opening a file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

INIDAT: tracer units for tracer = tracername must be in KG/KG

Ensure that any tracers being read from an initial conditions dataset are in these units.

INITAL: the regeneration file size is too small: Maximum file size (mxszrg): *nn* Size of a latitude record: *mm*

The maximum regeneration file size as defined by the namelist variable `mxszrg` is declared too small (default size 380 Mb). The user must set the `MXSZRG` namelist variable to a larger size (size of a latitude record).

INITAL: too many primary [secondary] regeneration files required:
Maximum number of files (pnrg1[2]) = *ii*
Maximum size of files (mxszrg) = *jj*
Actual number of files required = *kk*
Actual size of latitude record = *ll*
Number of latitudes per file = *mm*

Either the parameter `pnrg1[pnrg2]` in `/comlun/` must be increased to the actual number of files required, or the maximum size of each regeneration file (`MXSZRG`) must be increased. `MXSZRG` may be set via the namelist.

INITCOM:mmax=ptrm+1 .gt. plon/2

The spectral resolution parameters are inconsistent.

INITEXT: Att length of from_hires is too long

Attribute length of initial dataset netCDF variable `from_hires` cannot be creater than 256

INITEXT: Failed to open file *filename*

I/O error while opening a file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

INITEXT: Initial sicthk doesnt match oro

The sea ice thickness data on the initial boundary dataset is inconsistent with the ice points defined by the `oro` field also on the initial boundary dataset. The user should verify that the initial boundary dataset is correct.

INITINDXF: ending advected tracer index *ii* not equal to pcnst = *jj*

total number of advected tracers must equal `pcnst`. Recheck namelist settings for `NSUR_ADV` and `pcsnt` setting.

INITINDXF: ending non-advected tracer index *ii* not equal to pcnst+pnats = *jj*

total number of non-advected tracers must equal `pnats`. Recheck namelist settings for `NSUR_NAD` and `PNATS` setting.

INTHT: call to assign command failed

Call to external routine assign encountered a problem.

INTHT: Too many history files declared, max= *nn* To increase, change parameter ptapes.

`ptapes` defaults to 6, including the primary history file.

INTHT: Unable to open file sunit: ios= *errcode*

An error status was returned from trying to open unit `sunit` for direct access on the SSD.

IOSTOP: I/O ERROR Message

`iostop` uses system software to interpret I/O errors. Errors handled by `iostop` will be identified by subroutine name.

LAGYIN: Departure point out of bounds: jcen,icount,pgls= *ii jj kk*

This is an indication that the model is in the process of blowing up. Check that namelist variable `DTIME` is appropriate for the resolution.

LAGYIN: Error: ppdy .ne. 4

parameter `ppdy` in `parslt.h` must be set to 4 for the semi-Lagrangian code to run properly.

LANDTYP: ERRORS
landtyp errors: Last one is lsm surface type = nn is undef for lon,lat = ii jj

This is an internal error which should never occur.

LANDTYP: ERRORS
nn landtyp errors: Last one is olson surface type = nn is undef for lon,lat = ii jj

This is an internal error which should never occur.

landtyp error: lsm surface type = *nn* is not valid for lon,lat = *ii jj*

This is an internal error which should never occur.

landtyp error: fland = land. lsm = *nn* for lon,lat = *ii jj*

This is an internal error which should never occur.

`landtyp error: fland = ocean. lsm =` *`nn`* `for lon,lat =` *`ii jj`*

This is an internal error which should never occur.

`landwat error: lake =` *`ii`* `and wetland =` *`jj`* `are greater than 100 for lon,lat =` *`ii jj`*

Printed error should not be greater than round-off.

`LIMDX: Local work array DELI not dimensioned large enough. Increase local parameter pbpts to nn`

This is an internal error which should never occur.

LINEMSBC: WARNING: fixer for tracer *ii* exceeded rr for *jj* points at k,lat = *kk*,*ll* Avg/Worst = *ss*

The global conservation fixer is not allowed to modify constituent values by more than a specified threshold. This prevents the fixer from introducing unreasonable local fluctuations. However, global conservation is no longer maintained.

`LININTERP: Failed to find interp values`

The linear interpolation routine was unable to find input array values between which it could interpolate. This is an internal error which should never occur. If it does, a message to ccm-bugs@ucar.edu may be in order.

LININTERP WARNING: *rr* % of output grid will have to be extrapolated

The interpolated-to array overlaps the interpolated-from array. Thus, some percentage of the output array will be extrapolated. This is a non-fatal error.

`LININTERP: Point found without interp indices`

This is an internal error which should never occur. Contact ccm-bugs@ucar.edu

`LININTERP: Must have at least 2 input points for interpolation`

Input array to be linearly interpolated must have at least 2 points. Check input arrays.

`LININTERP: Non-monotonic coordinate array(s) found`

Coordinate arrays for interpolation must increase monotonically. Check input arrays.

`LIMDX: Local work array DELI not dimensioned large enough Increase local parameter pbpts to iiiii`

The *parameter* `pbpts`, local to subroutine `limdx`, must be increased to the value of `pidim`.

`lsmctli error: must set dtime`

LSM value for `DTIME` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nestep`

LSM value for `NESTEP` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nnbdat`

LSM value for `NNBDAT` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nnbsec`

LSM value for `NNBSEC` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nndbas`

LSM value for `NNDBAS` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nnsbas`

LSM value for `NNSBAS` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nrevsn in namelist for branch run`

LSM value for `NREVSN` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must set nsrest`

LSM value for `NSREST` obtained from atmospheric component is not correct. Check namelist input.

`lsmctli error: must specify case`

LSM value for `CASEID` obtained from atmospheric component is not correct. Check namelist input.

`LSMRDINIT: lonsiz=` *`ll`* `must equal lsmlon=` *`jj`*

Number of longitudes on LSM input data must equal number of LSM grid longitudes (see `lsmpar.h`).

`LSMRDINIT: latsiz=` *`ll`* `must equal lsmlat=` *`jj`*

Number of latitudes on LSM input data must equal number of LSM grid latitudes (see `lsmpar.h`).

`LSMRDINIT: levsiz=` ***`ll`*** **`must equal msl=`** ***`jj`***

Number of soil layers on LSM input data must equal number of LSM soil layers (see `lsmpar.h`).

`LSMRDINIT: bassiz=` ***`ll`*** **`must equal ndrnmax=`** ***`jj`***

Maximum number of drainage basins on LSM input dataset must equal number set in `basinpar.h`.

`(LSMRDINIT): time invariant data must be ordered lon, lat, time`

Self explanatory

`(LSMRDINIT): drngarea data must be ordered lon, lat, time`

Self explanatory

`(LSMRDINIT): drngarea data must be ordered basin, lon, lat, time`

Self explanatory

`(LSMRDINIT): incompatibility between calculated and input kpt`
`input kpt =` ***`kk`***
`calculated kpt =` ***`ll`***

Number of subgrid points on initial LSM dataset does not match that expected by LSM

`LSMTCI fatal error: discrepancy in subgrid points`
expecting *ii* points but found *jj*

Internal error in the LSM consistency check for the number of subgrid points. This error should never occur.

`LSMTCI fatal error: weights do not sum to 1`
`lon =` ***`ii`*** **`lat =`** ***`jj`*** **`:sum =`** ***`kk`***

Internal error. The LSM subgrid weights should sum to 1 for each land cell. This error should never occur.

`LUNITS: call to assign cmd failed`
`LUNITS:Unable to open file` ***`uu`*** **`: ios=`** ***`errcode`***

Error status *`errcode`* was returned when opening unit *uu* for direct access.

`map_i error srf = lsm: srf field not conserved`
`global sum lsm field =` ***`aa`***
`global sum srf field =` ***`bb`***

Dataset inconsistency. Should never occur if the supported datasets are used.

MCUTS: Workleft not zero. Value is *ii*

This is an internal error to the part of the SPMD implementation which parcels out spectral space work to individual processes. The error should never occur.

`msspni error: logname not defined`

LSM error in obtaining user logname from `LOGNAME` environment variable.

`msspni error: mass store file`

LSM could not define MSS pathname correctly.

`NAVU:Ran out of Fortran unit numbers`

Routine `navu` cannot find an unused logical unit number between 1 and 99.

NCDERR: ERROR in file *filename*: error

```
(orb_params) Have to specify orbital parameters!'
Either set: iyear_AD, or obliq, eccen, and mvelp:
iyear_AD is the year to simulate the orbit for  (ie. 1950):
obliq, eccen, mvelp specify the orbit directly:
The AMIP II settings (for a 1995 orbit) are:
obliq = 23.4441
eccen = 0.016715
mvelp = 102.7
```

The orbital parameters are required namelist items, you have to set either **`IYEAR_AD`** or the parameters: **`OBLIQ`**, **`ECCEN`**, and **`MVELP.`** See Table 2.5 or Example 4 for more details on the orbital parameters.

`(orb_params): Input obliquity unreasonable:`*`nn`*

The namelist value entered for obliquity is outside the reasonable range of the variable. Change your namelist and rerun. See the section on namelist variables for more details on obliquity.

`(orb_params): Input eccentricity unreasonable:`*`nn`*

The namelist value entered for eccentricity is outside the reasonable range of the variable. Change your namelist and rerun. See the section on namelist variables for more details on eccentricity.

`(orb_params): Input mvelp unreasonable:`*`nn`*

The namelist value entered for mvelp is outside the reasonable range of the variable. Change your namelist and rerun. See the section on namelist variables for more details on mvelp.

```
(orb_params) orbit only valid for years +-1000000
(orb_params) Relative to 1950 AD
(orb_params) # of years before 1950: nn
(orb_params) Year to simulate was : yyyy
```

The namelist variable IYEAR_AD can only be set to years that are within one million years of the year 1950 AD. The algorithm that calculates the earth±s orbital parameters is not accurate outside this time range, so the model will stop rather than trying to run with badly specified orbital characteristics. To get around this you need to calculate the orbital parameters for the year you want to simulate and enter those in the namelist rather than setting IYEAR_AD.

`(OPNFIL): local filename has zero length`

An input pathname must include an actual filename.

`(OPNFIL): failed to open file` *`filename`* `to unit` *`uu`* `ierr=`*`errcode`*

I/O error while opening a file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

`OUTFLD:Bad tens digit in mflds= t.`

Subroutine *`outfld`* encountered an invalid value in the tens digit of mflds(1). Valid values are 0 (instantaneous), 1 (averaged), 2 (minimum), and 3 (maximum). Check settings in subroutine `bldfld`.

`OZNINI: Data must be ordered lon, lev, lat, time`

Input data on the ozone netCDF boundary dataset are incorrectly ordered.

`OZNINI: Failed to find dates bracketing mcdate, mcsec=` *`yymmdd ss`*

The model integration time does not fall between available dates on the ozone dataset.

`OZNINI: levsiz=` *`levels`* `must = pozlev=` *`ozonelevels`*

The number of vertical levels on the ozone boundary dataset does not match that expected by the model. Solution: set cpp token POZLEV defined in params.h to the number specified by "levsiz" in the error message.

`OZNINI: When cycling ozone, dataset must have 12 consecutive months of data starting with Jan`
`Month` *`nn`* `of dataset says date=` *`yymmdd`*

This constraint only applies when namelist variable `ozncyc` is set to .true.

`OZNINI: When cycling ozone, dataset must have 12 consecutive months of data starting with Jan`
`Current dataset has only` *`nn`* `months`

This constraint only applies when namelist variable `ozncyc` is set to .true.

`OZNINT: Attempt to read past end of O3 dataset`

The model integration time does not fall between available dates on the ozone dataset.

OZNINT: Bad fact1 and/or fact2=fact1,fact2

Error interpolating the ozone dataset to model time. The interpolation factors do not agree to within 32 bit roundoff. This is an internal error which should never occur.

OZNINT: Input ozone for date *yymmdd* & sec second does not exceed model date xxxx sec yyy Stopping.

The model integration time does not fall between available dates on the ozone dataset

PACKAF:*WARNING*** This stub routine should not be called. NDENS should be set to 1. History data will NOT be packed.**

Packing is only available on an NCAR Cray. Solution: set namelist variable `NDENS` to 1.

POINTS:Mismatch between calculated length and parameter NLNBUF1 (or NLNBUF2) = *nn*, PFLENB (or PFLENA) = *mm*

The allocated space as indicated by parameter value `pflenb` or `pflena` is insufficient for the corresponding calculated length of data. Probable cause:

PREALC: Dummy routine for this machine architecture

Non-fatal. This disk-space pre-allocation routine only does useful work on Cray PVP architectures.

PUTFIL: Error from mswrite -- *errmsg*

Error message returned from library routine `mswrite`.

PUTFIL: Error from shell cmd

Self explanatory

rank: WARNING maximum test failed

LSM internal error. Should never occur.

rank error: i_1 = missing

LSM internal error. Should never occur.

RESUME: nstep(starting time step).ge.nstop. nstep = *nn* nstop = *oo*

Starting time step on restart or branch run is greater than ending timestep

QNEG3 from NNNNNN:m=*mm* lat=*ll* Min. mixing ratio violated at iiii points. Reset to *r.rrrrrEee* Worst = *w.wwwwwEee* at i,k=*ii kk*

This nonfatal message advises that some point values of the constituent field have fallen below their respective minimum acceptable threshold (see qmin(pcnst) in comqmin.h), and have been reset to the minimum threshold value. This should be a relatively rare occurrence, i.e., frequent occurrences of this message indicate a fundamental problem with the treatment of the particular constituent field.

QNEG4 WARNING from subroutine lat = *ii*; Max possible LH flx exceeded at *jj* points. Worst excess = *rr* at i = *kk*

This non-fatal message indicates that the model has computed a latent heat flux that will lead to negative water vapor mixing ratios in the lowest level of the atmosphere. The model will transfer this excess to the sensible heat flux term.

RADOZN: Bad ozone data: non-monotonicity suspected

Ozone boundary dataset may be corrupted.

RDHDR: integer (or character or real) header record too long. Allocated space =*nn*, Actual length= *mm*

Not enough space was allocated for one of the header records. If the actual read of the record produces an error, control returns to the calling routine for error processing.

READINITIAL: Failed to open file *filename*

I/O error while opening a file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

ERROR: READINITIAL model levels plev= *jj*, do not match levels in initial dataset mlev= *kk*

Pressure levels in initial dataset are inconsistent with the model.

READARR: I/O error
READARR: End of file

`readarr` calls `iostop` to document these errors.

READRG: I/O error in record number nn
READRG: End of file

`readrg` calls `iostop`to process errors encountered while reading the regeneration files.

READRG: Must change case name on a branch run. Previous name = *PrevCaseName*, current case name = *CurrCaseName*

To avoid overwriting existing output files, the case name (namelist variable CASEID) on a branch run cannot be the same as the case from which the branch is made.

READRIC: bad read of record *nn* from unit *uu*

Direct access *read* returned an error on unit *uu*. Routine `iostop` is called to further explain the error message.

RELAVU: unit *uu* is not flagged as in use

This non-fatal message indicates that an attempt was made to release a Fortran unit which was not already in use.

`RELUNIT (or RELAVU): Attempt to return out of range unit=`*`uu`*

Unit numbers must be in the range 1 to 99.

RELUNIT: Warning: Unit *uu* is not flagged as in use

This non-fatal error indicates that the user is trying to free a Fortran unit number that was never allocated by NAVU.

`RESTRT: Error returned from RDSIGAPB for record `*`nn`*

Restart and regeneration runs use information from the last history file header. An error was encountered in reading a header from this history file.

`restrt.ERROR: Cannot parse restart full pathname: `*`filename`*

Problem in restart file pathname. Restart pointer file may be hosed.

`RESTRT: No restart data available`

A restart dataset was not specified in the namelist of the previous run. Therefore, no regeneration datasets were written.

`RESTRT: too many secondary regeneration files required: write (6,*) Maximum number of files (pnrg2) = pnrg2 write (6,*) Maximum size of files (mxszrg) = mxszrg write (6,*) Actual number of files required = nnrg2 write (6,*) Actual size of latitude record = ltszrg write (6,*) Number of latitudes per file = nltrg`

Self explanatory.

`RESUME: nstep(starting time step).ge.nstop nstep = `*`nn`*`, nstop = `*`mm`*

namelist variable `NESTEP` should be greater than the restart timestep.

`RGNFLS (or RGNHBF):Failed to open file `*`filename`*`, ierr= `*`errcode`*

Subroutine `rgnfls or rgnhbf` was unable to open one of the regeneration datasets. The I/O status value is printed for more information. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

`RGNHBF: the regeneration file size is too small:`
`Maximum file size (mxszrg): `*`ii`*
`Size of a latitude record: `*`jj`*

The maximum regeneration file size as defined by the namelist variable `MXSZRG` is declared too small (default size 380Mb). The user must set the `MXSZRG` namelist variable to a larger size.

RGNHBF: too many hbuf regeneration files required:
Maximum number of files (pnrg1[2]) = *ii*
Maximum size of files (mxszrg) = *jj*
Actual number of files required = *kk*
Actual size of latitude record = *ll*
Number of latitudes per file = *mm*

parameter `pnrg1[pnrg2]` in `/comlun/` must be increased to the actual number of files required or the maximum size of each regeneration file (`MXSZRG`) must be increased. `MXSZRG` may be set via the namelist.

SCAN1BC: Failed to open file *filename* ierr= *errcode*

History tape *filename* could not be opened. Check file existence, and permissions.

The I/O status value is printed for more information.

SETTAU:UNSTABLE MEAN TEMPERATURE. STOP.

The mean temperatures set for the semi-implicit scheme result in an unstable scheme which would blow up if used.

SET99 -- INVALID N

Error in FFT library routine

SGHPHIS: Failed to open file *filename*

I/O error while opening the high resolution topo file. Make sure the correct filename (full filepath must be specified if the file is not in the local directory) is indicated by the namelist parameter `TOPOFILE`. This error is also indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.

SLTINT: Did not complete computations for all departure points

An internal error occurred while performing semi-Lagrangian interpolation.

SLTINT: ERROR: this routine does not provide shape preserving capability for horizontal-only interpolation

An internal error occurred while performing semi-Lagrangian interpolation.

SLTINT: ERROR: this routine does not provide shape preserving capability for vertical-only interpolation

An internal error occurred while performing semi-Lagrangian interpolation.

SLTINT: Error: must specify at least one of lhrzint or lvrtint to be true.

An internal error occurred while performing semi-Lagrangian interpolation.

SLTWGTS: Did not complete computations for all departure points

An internal error occurred while performing semi-Lagrangian interpolation.

soilcol error: bats soil color = *nn* is not valid for lon,lat = *ii jj*

This is an internal error which should never occur.

soilcol error: lsm soil color = *nn* is not valid for lon,lat = *ii jj*

This is an internal error which should never occur.

soiltex: WARNING maximum test failed at 10^6 and passed at 10^7

This is an internal error which should never occur.

soiltex error: did not find input cell with maximum overlap

This is an internal error which should never occur.

soiltex error: sand = *nn* silt = *mm* and clay = *oo* do not sum to 100 for lon,lat = *ii jj*

This is an internal error which should never occur.

soiltex error: sand = *nn* silt = *mm* and clay = *oo* not assigned to soil type for input grid lon,lat = *ii jj*

This is an internal error which should never occur.

soiltex error: sand = *nn* silt = *mm* and clay = *oo* not assigned to soil type for output grid lon,lat = *ii jj*

This is an internal error which should never occur.

SOMINI: Allocated space for som date variables is insufficient.
Please increase parameter totsomsz to at least *nn* and recompile.

Self-explanatory.

SOMINI: latsiz=*ii* must = plat=*jj*

Latitude dimension on SOM boundary dataset is inconsistent with model resolution.

SOMINI: Failed to find dates bracketing mcdate mcsec= *yymmdd* ss

The date values for the SOM boundary dataset are inconsistent with the model time. Check value of namelist variable NNBDAT.

SOMINI: lonsiz=*ii* must = plon=*jj*

Longitude dimension on SOM boundary dataset is inconsistent with model resolution.

SOMINI: When cycling som, 1st month must be 1

When running with the SOM and SSTCYC set to TRUE (i.e. using climatological mixed layer depths) the first month in the SOM dataset must correspond to January.

SOMINI: When cycling som, must have 12 consecutive months of data

When running with the SOM and SSTCYC set to TRUE (i.e. using climatological mixed layer depths) the SOM dataset must have 12 months worth of data to run from, with each of the months in consecutive order from January to December.

`SOMINT: Attempt to read past end of SOM dataset`

A SOM run attempted to read past the end of the SOM dataset. Typically this indicates the SOM dataset was too short for the type of run being done.

`SOMINT: Bad fact1 and/or fact2=fact1,fact2`

Error interpolating to a daily value. The interpolation factors do not agree to with in 32 bit roundoff.

`SOMQAD: iam=`*`p`*` nlats,plat=`*`nn,nn`*

Internal error in PVM routines in SOMQAD. Only applies to message passing version of CCM3

`SOMQAD: iam=`*`p`*` Do not have latitude `*`ll`*

Internal error in PVM routines in SOMQAD. Only applies to message passing version of CCM3

`SPHDEP: ****** MODEL IS BLOWING UP *********`
`Parcel associated with longitude `*`ii`*` level `*`kk`*` and latitude `*`jj`*` is outside the model domain.`

Routine SPHDEP has recognized unreasonable values for Model winds.

`(SPLITF): EOF reached in restart dataset`

There is a problem reading the restart dataset. It may be corrupted, or incompatible with model version being run.

`SRFTSB: Error returned from mtdlss`

An error has been detected in the multiple tri-diagonal linear system solver `mtdlss`. The error can be one of two types: a declared storage error or a numerical error (e.g., a singular matrix). Print the error status flag, `ier`, and see `mtdlss` for additional detail.

`SRFTSBI: error returned from mtdlss`

An error has been detected in the multiple tri-diagonal linear system solver `mtdlss`. The error can be one of two types: a declared storage error or a numerical error (e.g., a singular matrix). Print the error status flag, `ier`, and see `mtdlss` for additional detail.

`SSTINI: Allocated space for sst data is insufficient.`

Please increase parameter totsstsz to *nn*and recompile.

Self explanatory.

SSTINI: Failed to find dates bracketing mcdate, mcsec= *yymmdd ss*

The date values for the SST boundary dataset are inconsistent with the model time. Check value of namelist variable `NNBDAT`.

SSTINI: latsiz=*ii* must = plat=*jj*

The latitude dimension on the netCDF boundary dataset is inconsistent with the model.

SSTINI: lonsiz=*ii* must = plon=*jj*

The longitude dimension on the netCDF boundary dataset is inconsistent with the model.

SSTINI: ERROR
When cycling sst, sst dataset must have 12
consecutive months of data starting with Jan
Current dataset has only *nn* months

Check the SST boundary dataset.

SSTINI: ERROR
When cycling sst, sst dataset must have 12
consecutive months of data starting with Jan
Month *nn* of sst dataset is out of order

Check the SST boundary dataset.

SSTINT: Attempt to read past end of SST dataset

A SOM run attempted to read past the end of the SOM dataset. Typically this indicates the SOM dataset was too short for the type of run being done.

SSTINT: Bad fact1 and/or fact2=*rr,ss*

Error interpolating to a daily value. The interpolation factors do not agree to within 32 bit roundoff.

SSTINT: Input sst for date *yymmdd* sec *ss* does not exceed model date *mcdate* sec *mcsec* Stopping.

Check that model dates are consistent with SST boundary dataset.

STEPON: ***Running out of CPU time -- terminate***

Subroutine `stepon` has determined that there is not enough CPU time left on this job to reach the next history file write time. The logical flag `nlend` is set so the model will terminate normally.

SURPHY: sol rad error:nstep = *ii* vector = *jj* point =*kk* lsm point = *ll* inbalance = *rr* w/m2

This is an internal error produced by the LSM and indicates that an energy inbalance has occurred. This error should never occur. If it does, it indicates a major problem with the simulation.

SURPHY: sur flx error:nstep = *ii* vector = *jj* point =*kk* lsm point = *ll* inbalance = *rr* w/m2

This is an internal error produced by the LSM and indicates that an energy inbalance has occurred. This error should never occur. If it does, it indicates a major problem with the simulation.

SURPHY: lon rad error:nstep = *ii* vector = *jj* point =*kk* lsm point = *ll* inbalance = *rr* w/m2

This is an internal error produced by the LSM and indicates that an energy inbalance has occurred. This error should never occur. If it does, it indicates a major problem with the simulation.

SURPHY: soi tem error:nstep = *ii* vector = *jj* point =*kk* lsm point = *ll* inbalance = *rr* w/m2

This is an internal error produced by the LSM and indicates that an energy inbalance has occurred. This error should never occur. If it does, it indicates a major problem with the simulation.

`TPHYSIDL: should not run model with "idlflag" set to "3" (experimental code). The valid options are 1 or 2.`

This is an internal error which should never occur.

`TPHYSIDL: flag for choosing desired type of idealized physics ("idlflag") is set incorrectly. The valid options are 1 or 2. idlflag is currently set to:` ***i***

This is an internal error which should never occur.

`TRAJDP: ****** MODEL IS BLOWING UP *********`
`Parcel associated with longitude` ***ii*** **`level`** ***kk*** **`and latitude`** ***jj*** **`is outside the model domain.`**

Unreasonable model winds have been detected.

`TREMAIN: Dummy for Cray system routine. Returning`

This routine is not available on non-Cray architectures.

`TRUNC: Error in truncation parameters ntrm.lt.(ptrk-ptrn)`
`TRUNC: Error in truncation parameters ptrk.lt.ptrn`

The horizontal truncation parameters in `pmgrid.h` are not consistent.

`UNPKAF: This routine should not be called--not implemented on this machine architecture`

Packing and unpacking is available only on an NCAR Cray. Solution: ensure that all boundary datasets are unpacked.

VERH2O: tot wat error: nstep = *ii* vector = *jj* point = *kk* lsm point = *ll* inbalance = rr *mm*

This is an internal error produced by the LSM and indicates that mass conservation has been violated. This error should never occur. If it does, it indicates a major problem with the simulation.

`VRTMAP: Not enough artificial grid intervals. Currently, "pmap" is set to` *pp*
`Reset parameter "pmap" to at least` *nn*

In the semi-Lagrangian transport scheme, the artificial vertical grid that maps departure points into the model grid is not large enough to handle the vertical resolution of this run. The user must increase the value of `stepon` local parameter `pmap`.

`WRAPUP: Failed to open file` *filename* `ierr=`*errcode*

Auxiliary files are opened to append by `wrapup`. This open failed. The I/O status value is printed for more information.

`WRITERIC: bad write of record` *nn* `to unit` *uu*

The Fortran direct access *write* to unit *uu* returned an error. `iostop` is called to further explain the error message.

`WRTARR:I/O error`

`iostop` is called to explain an error returned from an array *write*.

`WRTRS1: Failed to open file` *filename*

I/O error while opening a regeneration file for writing. Usually indicative of a problem with the file system. User should make sure they haven't surpassed any local disk limits and that the file and directory permissions are correct.