User's Guide to NCAR CCM3

Thomas L. Acker
Lawrence E. Buja
James M. Rosinski
John E. Truesdale

CLIMATE AND GLOBAL DYNAMICS DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO
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 Mariana Vertenstein, Brian Eaton, Dave Williamson, and Jerry Olson.

A significant portion of the material contained herein was extracted from the user's manual for the previous model, version CCM2. Thus, credit needs to be given to previous authors Linda Bath and Jerry Olson for their contributions.
Table of Contents

Acknowledgments .................................................................................................................. iii
List of Figures ........................................................................................................................... ix
List of Tables ........................................................................................................................... xi
How To Use This Guide .......................................................................................................... xiii

I. Introduction ............................................................................................................................ 1
   A. Historical Development of the CCM3 .............................................................................. 1
   B. Overview of Running CCM3 .......................................................................................... 3
   C. Overview of the Structure of CCM3 .............................................................................. 4
   D. Summary ......................................................................................................................... 6

II. Using CCM3 .......................................................................................................................... 7
   A. How to Build and Run CCM3 ......................................................................................... 9
      1. Interactively Creating and Running CCM3 ............................................................... 9
         (a) Interactively Creating the Executable ..................................................................... 9
         (b) Interactively Running the Executable .................................................................. 12
      2. Creating and Running the Executable in Batch Mode ............................................. 13
      3. Message Passing Using PVM ................................................................................ 16
   B. Model Input Variables - Namelist Input ....................................................................... 19
   C. Initialization and Continuation Runs .......................................................................... 27
      1. Definitions .................................................................................................................. 27
      2. Example Runs: Setting the Input Variables ............................................................... 28
         (a) Example 1 — Initial Run, One Day ..................................................................... 29
         (b) Example 2 — Initial Run, T21 on a SUN or RS-6000 ........................................ 32
         (c) Example 3 — Restart Run, 30 Days .................................................................... 34
         (d) Example 4 — Regeneration Run, Retrieve Lost File ......................................... 36
         (e) Example 5 — Branch Run .................................................................................. 37
         (f) Example 6 — Initial Run, Add Fields to History File ........................................ 39
         (g) Example 7 — Run with Multiyear SST Dataset ................................................. 40
         (h) Example 8 — Generate a Monthly-Averaged History File ................................ 42
         (i) Example 9 — Debugging Run .............................................................................. 43
   D. Model Input Datasets ...................................................................................................... 45
1. Initial Dataset ................................................................. 45
2. Boundary and Prescribed-Value Datasets ................................................................. 46
   (a) Time-Invariant Boundary Dataset ........................................................................... 47
   (b) Time-Variant Sea-Surface Temperature (SST) Dataset ........................................... 47
   (c) Time-Variant Ozone, Prescribed-Value Dataset .................................................... 47
3. Land Surface Model (LSM) Datasets ....................................................................... 48
   (a) FSURDAT: The required time-invariant LSM boundary dataset ......................... 48
   (b) FINIDAT: The optional LSM initial dataset ............................................................ 49
4. Slab Ocean Model (SOM) Datasets ........................................................................... 49
   (a) NCDATA: The SOM initial dataset ........................................................................... 49
   (b) TVBDS: The time-varying SOM boundary dataset .................................................. 49

E. Model Output Datasets ......................................................................................... 51
1. Model History Files .................................................................................................. 51
   (a) Primary History File ................................................................................................. 52
   (b) Auxiliary History Files ............................................................................................ 61
   (c) Naming the Primary and Auxiliary History Files ..................................................... 62
2. Restart and Regeneration Datasets ......................................................................... 63
   (a) Types of Restart and Regeneration Files .................................................................. 63
   (b) Naming of Restart and Regeneration Files ............................................................... 64
3. Mass Store Archiving of History and Restart/Regeneration Files ......................... 67

F. Model Printed Output .............................................................................................. 69
1. Printout from a CCM3 Run ....................................................................................... 69
2. Model Error Messages ............................................................................................... 74

G. Using the Slab Ocean Model (SOM) ...................................................................... 91

H. Getting Help ........................................................................................................ 93
   (a) The CCM Web Page ................................................................................................. 93
   (b) The ccm-users e-mail Group .................................................................................. 93

III. CCM3 Internals ..................................................................................................... 95
A. Design Philosophy of CCM3 .................................................................................. 95
B. Model Code Flow .................................................................................................. 99
   1. Initialization .............................................................................................................. 99
   2. Time Integration ..................................................................................................... 99
      (a) Latitude Scans ...................................................................................................... 100
      (b) Spectral Space Computations ............................................................................. 101
      (c) Computational Driving Routines for Each Latitude Band .................................. 102
C. Data Structures ................................................................................................... 103
1. Vertical Coordinate ................................................................................................. 103
2. Grid-point Data Structures ....................................................................................... 103
   (a) Model Buffer ................................................................................................... 106
   (b) History Buffer ................................................................................................. 107
   (c) Absorptivity/Emissivity Arrays ...................................................................... 108
   (d) Out-of-Core Data Storage: The SSD Work Units .......................................... 108
   (e) Local Workspace ............................................................................................ 109
   (f) In-Core Grid-point Arrays .............................................................................. 110
3. Spectral Data Structures ........................................................................................... 112
4. Other Common Blocks and Header Files: The Parameterization Interface ............. 112

D. CCM3 Multitasking Strategy .................................................................................. 115
1. Shared-Memory Multitasking .................................................................................. 115
   (a) Shared-Memory Management ........................................................................ 117
2. Distributed Memory Multitasking............................................................................ 118
   (a) Distributed Memory Management .................................................................. 119
   (b) Distributed Memory I/O ................................................................................. 119

E. Details of CCM3 Input and Output Datasets ..................................................... 121
1. History Files ............................................................................................................. 121
   (a) History File Format ......................................................................................... 121
   (b) History File Header Records .......................................................................... 123
   (c) Latitude Data Records .................................................................................... 129
   (d) History Buffer ................................................................................................. 132
   (e) History File Code Flow .................................................................................... 136
2. Initial and Boundary Condition Datasets .................................................................. 138
3. Restart/Regeneration Files ....................................................................................... 138
   (a) Restart and Regeneration File Contents .......................................................... 140
   (b) Restart and Regeneration Code Flow ............................................................. 141

F. CCM3 Disk and Mass Store File Management .................................................. 145
1. CCM3 Disk File Management on an NCAR Cray .................................................. 148
2. Use of the NCAR Mass Storage System .................................................................. 149
3. Issuing Shell Commands from the Model ........................................................... 149

IV. Changing the Model .............................................................................................. 151
A. Modifying the Code .............................................................................................. 151
B. Adding New Variables ......................................................................................... 154
C. Adding Constituents to the Model: The Semi-Lagrangian Transport ........ 156
   1. Initializing New Constituent Data Fields ............................................................. 157

Table of Contents
2. Surface Fluxes, Sources, and Sinks ................................................................. 158
3. Adjustment Processes ...................................................................................... 159
4. Global Minima ................................................................................................. 159

D. Changing Resolution ....................................................................................... 160

E. Adding Parameterizations ............................................................................... 160

F. Trouble-Shooting Model Changes .................................................................. 161
   1. Resource Allocation Errors .......................................................................... 161
   2. Coding Errors ............................................................................................. 162
   3. Formulation Errors ..................................................................................... 164

G. CCM3 Coding Standard ................................................................................... 164
   1. Code Appearance ....................................................................................... 164
   2. Other Fortran Considerations ..................................................................... 166
   3. Program Routine Names and Variable Names ........................................... 167
   4. Code Structure ........................................................................................... 167

References .......................................................................................................... 169

Appendix A: Glossary .......................................................................................... 171

Appendix B: CCM3 Calling Tree .......................................................................... 175

Appendix C: CCM3 Printed Output ...................................................................... 187
List of Figures

Figure I.1: Conceptual flowchart of the CCM3 code.................................................................5

Figure III.1: CCM3 Hybrid Vertical Coordinate.................................................................104

Figure III.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.......................... 107

Figure III.3: Use of SSD Work Units in the CCM3 out-of-core configuration. Arrows indicate data movement between Model buffers (vertical rectangles) and the out-of-core work 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 labelled with the subroutine name responsible for the data transfer............................................................. 109

Figure III.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 code................................................................. 110

Figure III.5: Spectral storage arrangement when architecture type is not CRAY. Dashed lines depict mapping of 2-dimensional conceptual location in triangular data structure to actual linear storage in memory................................................................................................. 113

Figure III.6: Spectral storage arrangement when architecture type is CRAY. Dashed lines depict mapping of 2-dimensional conceptual location in triangular data structure to actual linear storage in memory................................................................................................. 114

Figure III.7: History file format. One disk history file contains mfilt time samples. Each time sample consists of three header records and plat latitude data records......................................................... 122

Figure III.8: A Typical history file latitude record of unpacked data, and its relationship to the pointers in mfllds and variable names in mcfllds................................................................. 131

Figure III.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. The lengths of the packed and unpacked latitude data records for the primary and first auxiliary history files are shown by nplen and nrlen, respectively........... 133

Figure B.1.a: Calling tree: Initializations, part (a)................................................................. 176

Figure B.1.b: Calling tree: Initializations, part (b)................................................................. 177
Figure B.2: Calling tree: Time-stepping loop ................................................................. 178

Figure B.3.a: Calling tree: Pre-coupler Gaussian latitude scan, part (a) ....................... 179

Figure B.3.b: Calling tree: Pre-coupler Gaussian latitude scan, part (b) ....................... 180

Figure B.4: Calling tree: Ocean and sea ice models ...................................................... 181

Figure B.5: Calling tree: Land surface model ................................................................. 182

Figure B.6: Calling tree: Post-coupler Gaussian latitude scan ...................................... 183

Figure B.7: Calling tree: Semi-Lagrangian transport .................................................... 184

Figure B.8: Calling tree: Spectral routines and conversion to grid-point space ............. 185
List of Tables

Table II.1: CCMEXP namelist Input Variables .................................................................20
Table II.2: LSMEXP namelist Input Variables .................................................................25
Table II.3: Initial Dataset Fields..........................................................................................46
Table II.4: Time-Invariant Boundary Dataset Field............................................................47
Table II.5: Land Surface Model Boundary Dataset Fields..................................................48
Table II.6: Land Surface Model Initial Dataset Fields.......................................................49
Table II.7: Slab Ocean Model Initial Dataset Fields...........................................................50
Table II.8: Slab Ocean Model Time-Varying Boundary Dataset Fields...............................50
Table II.9: CCM3 Master Field List....................................................................................53
Table II.10: Additional Fields on a Monthly-Averaged Primary History File......................61
Table III.1: History File Header Record 1—Integer Variables ........................................123
Table III.2: History File Header Record 2—Character Variables* .....................................126
Table III.3: History File Header Record 3—Real Variables ...............................................128
Table III.4: Description of Field Information List Arrays, MFLDS and MCFLDS................130
Table III.5: Input Dataset Header Fields That Must be Set ..............................................139
Table III.6: System Routine Calls in CCM3......................................................................147
Table IV.1: SI Units of History File Fields........................................................................155
How To Use This Guide

This document is intended to be used both by the new user who will be running CCM3 for the first time, and also by the advanced user who may wish to make changes to the code. Sections of this report relevant to the new user are Section I, "Introduction," and Section II, "Using CCM3." These sections are intended to familiarize the user with the overall setup of CCM3, as well as explain how to obtain, compile, load, and execute the code. Section III, "CCM3 Internals," and Section IV, "Changing the Model," are provided for the advanced user interested in modifying the source code.

In preparing this User’s Guide, the authors have adopted certain conventions to make repeated explanation of CCM3 variables, program modules, etc., unnecessary. The word “Model” with an initial capital letter refers to the standard CCM3 version of the NCAR Community Climate Model. Certain common English terms are used in unique ways when describing the CCM. These terms, plus commonly-used acronyms, appear with definitions in "Appendix A: Glossary."

In order to make the text more readable, certain font conventions have been adopted. These conventions are summarized below:

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| In CCM3, there is nearly a one-to-one correspondence between procedure name and filename. For instance, subroutine stepon is stored in file stepon.F. Thus, to aid in distinguishing procedure names form variable names, procedure names are in lower-case courier font, followed by a " .F" or a " .h", whichever is appropriate. | stepon.F  
comhst.h |
| Fortran keywords and C preprocessor keywords are in italics.                | namelist  
common  
real  
#include |
| All namelist variables, C preprocessor tokens, initial dataset fields, and history file header-record variables are in upper-case courier font. | NSTEP  
NINAVG(i)  
ORO |
| Non-namelist and non-header-record program variables are in lower-case courier font. | lhbuff  
hstwrt(itape) |
| System commands are in bold courier font.                                  | mswrite  
tar xvf |
I. Introduction

This document is the User's Guide to version 3 of the NCAR Community Climate Model, denoted CCM3. It is a companion document to Description of the NCAR Community Climate Model (CCM3) (Kiehl et al., 1996), which describes the governing equations, numerical algorithms, and physical parameterizations used in CCM3. The CCM3 is copyrighted by the University Corporation for Atmospheric Research.

This Guide was written with two goals in mind: (1) to provide detailed information that describes how to use CCM3; and (2) to furnish the user interested in modifying the CCM3 code with an in-depth description of the code organization, data structures, and other important details. The form of this report is similar to the User's Guide to NCAR CCM2 (Bath et al., 1992), but has been reorganized and updated to reflect the changes made in creating CCM3.

Section II of this report, “Using CCM3”, starts by delineating the steps involved in obtaining, compiling and executing CCM3. Next, the namelist input that is used to control a run is described in detail, followed by several examples. The input and output datasets are then discussed, along with a description of the Model printed output and error messages. The section concludes with a brief description of the Slab Ocean Model followed by information on how to get help with the CCM.

For the user interested in making changes to CCM3 code, Section III, "CCM3 Internals," has been provided. This section illuminates the basic structure and fundamental features of the CCM3 code. Section IV, "Changing the Model," addresses issues relevant to modifying code, such as the CCM3 coding convention, and includes some specific examples.

A. Historical Development of the CCM3

The original versions of the NCAR Community Climate Model were based on atmospheric general circulation models developed at two major modeling centers. Version CCM0A had its origins in an early version of the Australian spectral model from the Australian Numerical Meteorology Research Center. CCM0B evolved from an early adiabatic, inviscid version of the spectral model developed at the European Centre for Medium Range Weather Forecasts (ECMWF). The physical parameterizations incorporated into these models included radiation and cloud routines developed at NCAR, as well as convective adjustment, stable condensation, vertical diffusion, surface flux, and surface-energy-balance prescriptions developed at the Geophysical Fluid Dynamics Laboratory (GFDL).

The second generation model, CCM1, evolved from the CCM0B version, and included significant improvements to the radiation and vertical finite-
Introduction

difference schemes. There were more modest improvements to the horizontal and vertical diffusion processes, surface energy exchanges, and the introduction of several new modeling capabilities, such as a seasonal mode in which the specified ocean surface conditions vary with time.

In CCM2, the CCM Core Group incorporated major changes to all aspects of the parameterized physics, changes in resolved dynamics, and introduced new modeling capabilities. In addition to these changes in formulation, the CCM Core Group entirely reimplemented the code, with three major objectives: greater ease of use and modification; conformation to a plug-compatible physics interface standard (defined in “Other Common Blocks and Header Files: The Parameterization Interface” on page 112); and the incorporation of single-job multitasking capabilities. These objectives were maintained in moving from CCM2 to CCM3.

CCM3 is the latest version of the NCAR Community Climate Model. It is a direct descendant of CCM2, and therefore contains much of the same code and uses a similar approach to solving the equations describing the atmospheric general circulation, as well as the physical parameterizations. The major modifications made in producing CCM3 are as follows:

- Clear-sky radiation:
  > Incorporation of trace gases in longwave radiation - CH₄, N₂O, CFC₁₁, CFC₁₂, minor CO₂ bands
  > Incorporation of background aerosol
- All-sky radiation:
  > Modified diagnosis of cloud optical properties (effective radius and liquid water path)
  > Incorporation of radiative properties of ice clouds
  > Modifications to cloud fraction parameterization (new convective cloud scheme and modified layered cloud scheme)
- Hydrologic cycle:
  > Modified non-local Atmospheric Boundary Layer - diagnosis of boundary layer height
  > Evaporation of stratiform precipitation
  > LSM land surface model (Bonan, 1996)
  > Dynamically determined z₀, the roughness height, over ocean
  > Incorporation of deep moist convection formalism - Zhang-McFarlane hybrid (see Kiehl et al, 1996)
- Slab ocean model (SOM) option
- Split scanl.F routine into scan1bc.F and scanlac.F in order to include flux-coupling capability
- Modified code to run on platforms other than the Cray Y-MP
- Message passing implementation
- Optimized code for improved execution rate, a leaner buffer model that uses less scratch I/O, and more flexible memory management options.
B. Overview of Running CCM3

CCM3 can be obtained as a compressed tar file via ftp. Once uncompressed and untarred, a c-shell script file can be run in order to configure the code on a given computer architecture at a given resolution. A "Makefile" is then processed by the make utility to compile the Model and generate an executable. Due to the complexity of the code, several steps are carried out in order to compile the code. However, because these steps are delineated in the Makefile, they are transparent to the user. The code can be run single-threaded or on multiple processors, and has been vectorized to ensure a fast rate of execution on Cray machines.

The CCM3 code, as its predecessors, is written in Fortran 77 with a few commonly supported Cray extensions. For example, loops are written with the do...end do structure, imbedded comments are included on many lines, and Cray pointers are used extensively. In order to make the code more readable and to permit it to be run on multiple computer architectures, preprocessor directives (cpp) are used at the beginning and throughout most subroutines (e.g. #include, #define, etc.).

Because of the preprocessor directives, the entire code must be run through the C preprocessor prior to compiling. CCM3 code also contains some directives to enable multitasking and vectorization. The multitasking directives (of the form CMIC$ ...) require preprocessing by a Fortran mid-compiler if the code is to be run on a Cray PVP (Parallel Vector Processor) machine using multiple processors. Compiler directives (of the form CDIR$ ...) are processed by the Fortran compiler. If run single-threaded or on a non-Cray machine, the multitasking and compiler directives are interpreted as Fortran comments.

CCM3 is composed of a large number of program files which are compiled and loaded into an executable file. In fact, each subroutine is contained within its own file (with very few exceptions). Model source code files are denoted by a " . F" (Fortran code) or a " . c" (C code) suffix. The capital "F" suffix indicates that the Fortran source contains preprocessing directives. The remaining files are named with a " . h" suffix. These files are header files that contain variable declarations, parameter definitions, and common block definitions. The contents of select header files are included via #include statements at the beginning (header) portion of each of the " . F" files.

As mentioned previously, the make utility processes instructions contained within the Makefile in order to compile the source code and generate an executable file. These instructions direct make to check which source code files need to be compiled, pass the code through the C preprocessor, then compile Fortran code using a Fortran 77 compiler (the Fortran mid-compiler is invoked as a command line option of the Cray cf77 Fortran compiler), or compile C code with an ANSI C compiler. The compiler
generates an object file (".o" suffix) corresponding to each source code file. For example, compiling the routine stepon.F will generate an object file with the name stepon.o. A loader is then used to gather all the compiled code into an executable file, including library ".o" files.

Useful references to aid in reading the code and understanding the Makefile are the *CF77 Fortran Language Reference Manual* (Cray, 1993), the *UNICOS User Commands Reference Manual* (Cray, 1994), *CF77 Commands and Directives* (Cray, 1993), and the local *man* page describing the *make* utility (or *GNU Make* (Stallman et al., 1991), if *gnumake* is available).

Types of CCM3 runs

Once an executable file has been made, there are two primary modes in which CCM3 may be run: an initialization run or a continuation run. An initialization run starts from a given set of initial and boundary condition datasets and executes for a user-defined number of time steps. Alternatively, a continuation run begins by using the output from a previous run of the Model and then executes for a user-defined number of time steps. During a run, output is generated in the form of history files, regeneration files, and printed output. Details of the input files, output files, and modes of operation are presented in Sec. II.

C. Overview of the Structure of CCM3

The basic structure of the CCM3 code can be separated into an initialization phase and a time-stepping phase. A conceptual flowchart is shown in Figure I.1. Along with a brief description of what happens within each phase are the names of routines responsible for controlling that aspect of the Model. For example, the time-stepping is controlled from within routine stepon.F, as indicated in the figure. This figure also serves as a reference linking key elements of the CCM3 physics (Kiehl et al., 1996) to the code. A comprehensive flowchart of the CCM3 calling tree, showing all program modules except single-purpose utility routines, is provided in "Appendix B: CCM3 Calling Tree".

Referring to Figure I.1, 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-stepping phase.

The time-stepping phase is where virtually all of the calculations are performed in order to solve the equations describing the atmospheric general circulation. As can be seen from Figure I.1, this phase is split into seven categories. These include a pre-coupler latitude scan, coupling to the land, sea, and ice computations, a post-coupler latitude scan, disposing files to the NCAR Mass Storage System, semi-Lagrangian transport, spectral space calculations, and a final latitude scan to convert between spectral
Figure 1.1 Conceptual flowchart of the CCM3 code.
space and grid-point space. After these seven steps have been completed, the time step is incremented and the steps are repeated. Note, the location within the code of certain important features, such as the tendency physics and the adjustment physics, have been included in the figure.

D. Summary

The CCM3 is the latest in a line of NCAR computer codes used for climate prediction. It has been written in Fortran 77 with some common extensions, and can be run on several different computer architectures at various resolutions. A fairly simple Makefile interface is provided to the user for building and compiling the code. The model is composed of an initialization phase and a time-stepping phase in which most of the calculations are performed. In the sections that follow, the particulars describing how to use the model, the internal structure of the code, and how to change the code are presented.
II. Using CCM3

This section provides information necessary to use the CCM3 version of the NCAR Community Climate Model. Detailed instructions are provided that inform the user how to obtain a copy of CCM3, then how to compile and execute either interactively or in batch mode. Following these instructions are several subsections that describe the required CCM3 input and resulting output. The user is assumed to be familiar with the Fortran programming language and to have a basic knowledge of the UNIX operating system.

Section II.A, "How to Build and Run CCM3," describes the steps required to obtain a copy of CCM3 via ftp, and how to set it up on various computer systems. The various architectures upon which the Model may be compiled and executed are discussed, along with the supported resolutions. Instructions demonstrating how to compile and execute are provided.

While the information furnished in Sec. II.A is intended to supply the user with the basic knowledge required to obtain the Model and run it, Section II.B through II.F presents many details about the input and output that permit the user to fully customize a run. Numerous aspects of a run are controllable by the user, such as what input datasets are used, what fields are output to a history file, the time step, etc. These features are controlled using a Fortran namelist, described in detail in Sec. II.B. This section is followed by several examples that demonstrate how to customize a run by setting the namelist variables. Section II.D then defines the contents of the initial and boundary datasets.

Output from the Model is generated in three basic forms: history files, regeneration files, and printed output. The history files are the principal form of output. These files contain grid-point data at various timesteps during a run. Regeneration files provide a "restart" feature to the Model, recording an image of the main model buffers. These files are intended to be read only by the Model when performing a continuation run and not to be viewed by the user. History and regeneration files are discussed in Sec. II.E. Model printed output is described in Sec. II.F, and is composed of all output directed to the standard output device. This includes an echo of the input namelist variables, compiler messages, Model statistics, and error messages.

CCM3 can be run with either the standard prescribed sea surface temperatures (with its implied sea ice distribution), or with an optional Slab Ocean Model (SOM) that computes sea surface temperature and sea ice distribution. A user familiar with running CCM3 using prescribed SST's will find using SOM very simple. Where appropriate in Sec. II.A through II.F,
Using CCM3

changes necessary to employ SOM are noted, but the main section describing how to run with SOM is Sec. II.G.

Information on getting help with the CCM is included in Sec. II.H. This section describes the ccm-users e-mail group, and how to find documentation available on the CCM web page.
A. How to Build and Run CCM3

CCM3 code, as well as boundary datasets needed to run the Model, 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.code.tar.gz
get ccm3.datasets.T42.COS.tar
quit
```

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

Three methods for creating and running the CCM3 executable are presented in the following subsections. The first method describes the procedure in an interactive environment where message-passing is not enabled. For long climate simulations it is usually more convenient to use a batch facility. Thus, the second method presented utilizes the NQS batch facility on a Cray computer. The third procedure describes how to build the message-passing (PVM) implementation.

1. Interactively Creating and Running CCM3
(a) Interactively Creating the Executable

The file ccm3.code.tar.gz contains CCM3 code in addition to scripts to compile, load, and run the Model. This file is a compressed tar file which must first be uncompressed with the GNU gunzip utility, and then “untarred” as follows:

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

The tar xvf command extracts component files, creating a ccm3/ directory hierarchy containing CCM3 code and scripts.

The tar file containing the datasets is not compressed and can be un tarred with the following command:

```
tar xvf ccm3.datasets.T42.COS.tar
```

Next, the user must configure the code for the desired target architecture, resolution, multitasking option, and active or inactive ocean. A C-shell
Using CCM3

The output from a typical session using the setup script is shown on the following page (note that all user supplied input is preceded by >>). After answering the questions, a subdirectory will be created with an architecture and configuration-specific name (e.g. CRAY.t42.spectral.std/) which contains all the code required to build the CCM3 executable. To compile and load the Model, just cd into this directory and type make. The make utility will compile all CCM3 routines, create an executable

Model configurations

Currently supported architectures are CRAY, T3D, RS6K, and SUN. Note that the token SGI also appears in the code, but the correctness of a simulation generated on this platform has not yet been verified. RS6K refers to an IBM RS-6000 and “CRAY” refers to the parallel vector processing (PVP) machines built by Cray Research, Inc. (CRI). Even though the T3D is a CRI product, it is not a PVP machine. Examples of PVP machines are the Y-MP, C90, and J90.

T42 is the standard horizontal resolution, but T31, T21, and T5 are also supported, primarily for either paleoclimate or debugging work. These lower resolutions require less CPU time, core memory, and disk space to execute.

Supported ocean configurations include “standard” mode, in which climatological sea surface temperatures are used, the “slab ocean” option (see "Using the Slab Ocean Model (SOM)” on page 91 for further details), and "csm". The csm configuration enables the atmospheric model to be included as a component in a system of geophysical models in which two-way interaction is enabled through the use of a flux coupler (see The NCAR CSM Flux Coupler (Kauffman, 1996)).

If the target architecture is anything but CRAY, then distributed memory message-passing multitasking can optionally be enabled. This option uses the PVM3 message-passing library from Oak Ridge National Lab (ORNL). Documentation on how to build and use PVM3, as well as the source code itself, is available from netlib via the World Wide Web at http://www.netlib.org/liblist.html. Refer to “Message Passing Using PVM” on page 16 of this guide for information on how to run CCM3 using PVM.

Creating an executable with 'make'

The output from a typical session using the setup script is shown on the following page (note that all user supplied input is preceded by >>). After answering the questions, a subdirectory will be created with an architecture and configuration-specific name (e.g. CRAY.t42.spectral.std/) which contains all the code required to build the CCM3 executable. To compile and load the Model, just cd into this directory and type make. The make utility will compile all CCM3 routines, create an executable
>>setup
Version 1.0 of setup
This script creates a directory whose name is architecture-dependent, resolution-dependent, and configuration-dependent. It then installs appropriate CCM3 source code, Makefile, and other ancillary files into this directory, and puts sample namelists into the subdirectory 'run/'.

Configuration 'sld' (semi-Lagrangian dynamics) is not yet implemented. Boundary datasets are not necessarily available for all resolutions and configurations. For further information, please see the CCM3 home page at http://www.cgd.ucar.edu/cms/ccm3/, and the CCM3 User's Guide.

Abbreviations are as follows:
sld: semi-Lagrangian dynamics
std: standard CCM3 configuration--climatological SST's and sea ice
som: Slab Ocean Model
csm: Climate System Model (builds the CCM3 configured for coupling to active ocean and sea-ice models).

Architecture? Options are CRAY T3D RS6K SUN [CRAY]
>>CRAY
Architecture is CRAY
If this code is being built to run on one of the NCAR Cray machines, the Makefile will be constructed to use library versions of the ECMWF FFT code and mass store interface routines. If not, the slower version of the FFT code and dummy Fortran interfaces for the mass store routines will be used

Is the target architecture one of the NCAR Cray machines? [yn]
>>y
Resolution? Options are t42 t31 t21 t5 [t42]
>>t42
Resolution is t42
Type of run? Options are std som csm [std]
>>std
Type of run is std
Type of dynamics? Options are spectral sld [spectral]
>>spectral
Type of dynamics is spectral
Message-passing is not enabled
Sea surface temperatures and sea ice distribution are computed from a boundary dataset.

Building header files params.h and preproc.h, Makefile, and list of dependencies required by 'make' for CCM3 to be run on architecture CRAY at a horizontal resolution of t42...

Copying over example namelists for initial and restart runs. You will probably need to edit these before running the model depending upon architecture, resolution, etc.
Using CCM3 (named ccm3bin), and put it in a subdirectory named run/. When compiling the code, it is expected that the target architecture is the same as the machine on which the tar file was expanded, except for the T3D where it is assumed the user is on a Cray PVP machine (the compiler cannot be run on the T3D itself). If the architecture is SUN, then the default target operating system is Solaris. However, if the operating system is SunOS, then sunos must be specified as the target on the make command line (i.e. make sunos).

If multiple processors are available, a parallel make can be accomplished on a Cray machine through use of the macro or environment variable NPROC (e.g. make NPROC=2), or machines upon which GNU Make is available by using gnumake with the -j option. This should speed up the build process by a factor approaching the number of processors chosen. Caution should be exercised on Cray machines, because if NPROC is set to the number of physical processors, it is possible to exceed memory and/or number of process limits. The best value for NPROC is therefore implementation-dependent, and must be determined through trial and error.

(b) Interactively Running the Executable

If all has gone well up to this point, there should be an executable file by the name of ccm3bin in the subdirectory run/. In order to run the Model, appropriate boundary datasets must be in place. Some of these datasets are ASCII, and others are in either COS-blocked or IEEE binary format. If the target architecture is CRAY or T3D, the COS-blocked datasets are needed. Otherwise, the IEEE datasets are used. The datasets can all be found at the same anonymous ftp location as the CCM3 code, under the name ccm3.datasets.<resolution>.<binary format>.tar, where <resolution> is one of T42, T31, T21, or T5, and <binary format> is either COS or IEEE. The tar file containing these datasets needs to be run through tar xvf to extract the component files. The user should put these files in the same directory as the ccm3bin executable in order for the Model to be able to access them. See Sec. II.D for further details about the boundary datasets.

If the code is being built to run on one of the NCAR Cray machines, COS-blocked versions of the T42 boundary datasets are stored in the directory /ccm/CCM3/data. They can be used in place by setting certain namelist variables appropriately. See Sec. II.B for detailed information on namelist input.

Once the boundary datasets are in place, the Model is ready for execution except for the possible specification of the environment variable NCPUS. If the code is targeted to run multitasked on a Cray PVP machine, NCPUS declares the number of CPU’s to use in parallel (e.g. setenv NCPUS 8). Its value should not exceed the number of physical CPU’s on the machine.
Namelist files

If the code has been configured using the script setup, there should be two sample namelist files located in the same directory as the executable. For an initial run, nl.init will cause the Model to run for one simulated day. Since the namelist file is read from the standard input device (stdin), the way to invoke the executable is: ccm3bin < nl.init. After completion of this run, the integration can be continued for another day by using the sample namelist file nl.restart: ccm3bin < nl.restart. These two sample namelists are intended only as examples, and will probably require editing for real applications.

Batch script

2. Creating and Running the Executable in Batch Mode

The above discussion assumes that the user is both building and running CCM3 interactively. Since climate integrations are often run many simulated years, it may be more convenient in such cases to use a batch facility to complete the run. An example of a batch script designed for a Cray Y-MP using the batch facility NQS (Network Queuing System) is shown below. Following the script is a detailed explanation of its contents.

```bash
#!/bin/csh -fx
#QSUB -q prem
#QSUB -IT 2000
#QSUB -lM 11Mw
#QSUB -lQ 14mw
#QSUB -eo #
#
# Untar the CCM3 source code. In this example, the tar file is assumed to be in /usr/tmp/$LOGNAME, and already "gunzipped".
#
cd $TMPDIR
tar xvf /usr/tmp/$LOGNAME/ccm3.code.tar
#
# Set environment variables for batch execution of setup script.
#
setenv CCMARCH CRAY
setenv CCMRES t42
setenv CCMDYN spectral
setenv CMCTYPE std
setenv NCAR
cd ccm3
./setup || exit 1
#
# cd into the directory created by "setup", then compile # and load CCM3
#
cd $CCMARCH.$CCMRES.$CCMDYN.$CMCTYPE $|| exit 2
make NPROC=4 $|| exit 2
#
# Untar the boundary datasets into the working directory.
```
In this particular example, the standard CCM3 (T42, climatological sea surface temperatures, spectral dynamics) runs out-of-core (i.e., uses the solid-state storage device SSD) for one simulated day. Note that the setup script is invoked in a batch fashion. That is, environment variables are set such that the script need not ask questions which require an interactive answer. In this example it is assumed that the file ccm3.code.tar, and the datasets ccm3.datasets.T42.COS.tar have been placed into the directory /usr/tmp/$LOGNAME prior to execution of the batch script (the user may have to create this directory—it is not provided automatically). This sample script (runmodel.example) is available in the subdirectory ccm3/CRAY from where ccm3.code.tar is untarred. The script can be submitted to NQS with the command:

qsub runmodel.example.

The following discussion provides detailed information about key sections in the CCM3 run script. A basic knowledge of the UNICOS operating system and the NCAR Cray environment is assumed. Refer to UNICOS User Commands Reference Manual (Cray, 1994) for more details. The namelist How to Build and Run CCM3
variables are explained in detail in “Model Input Variables - Namelist Input” on page 19.

The "QSUB" directives at the top of the script specify maximum CPU time (-1T), job class (-q), maximum main memory (-1M), and maximum SSD storage (-1Q) for the job.

```
#QSUB -q prem
#QSUB -IT 2000
#QSUB -1M 11Mw
#QSUB -1Q 14mw
#QSUB -eo
```

Job class, premium
CPU time limit for the job, 2000 seconds
Main memory requirement, 11 megawords
SSD requirement, 14 megawords
Directs stdout and stderr to the same file

These memory and SSD limits are sufficient for running the CCM3 at T42 horizontal resolution multitasked on eight processors. Refer to “Shared-Memory Management” on page 117 for more information on memory requirements.

```
setenv CCMARCH CRAY
setenv CCMRES t42
setenv CCMDYN spectral
setenv CCMTYPE std
setenv NCAR
```

$CCMARCH Target architecture. CRAY refers to shared-memory parallel vector processor (PVP) machines built by Cray Research, Inc. The Cray Y-MP, C90, and J90 are all PVP machines. Supported resolutions are CRAY, SUN, RS6K, T3D.

$CCMRES Horizontal resolution. Supported resolutions are t42, t31, t21, and t5.

$CCMTYPE Type of run. std means climatological sea surface temperatures will be used. If CCMTYPE = som, the slab ocean option is enabled. CCMTYPE = csm configures the code for coupling to active ocean and sea ice models. For further information, see The NCAR CSM Flux Coupler (Kauffman, 1996).

$NCAR If set, this environment variable declares that the target Cray is located at NCAR. This means that site-specific library routines will be used, such as those which access the NCAR Mass Storage System, and assembly language versions of the FFT package.

```
setenv NCPUS 8
```
Using CCM3

$NCPUS A system environment variable specifying the maximum number of CPU’s that the Model will use in parallel.

Assign scratch files

| assign -F sds.scr.novfl u:11 |
| assign -F sds.scr.novfl u:21 |
| assign -F sds.scr.novfl u:22 |
| assign -F sds.scr.novfl u:60 |
| assign -F sds.scr.novfl u:70 |

These assign commands declare that I/O to Fortran unit numbers 11, 21, 22, 60, and 70 will be done using Secondary Data Segments associated with the solid state disk (SSD). These units represent scratch work space used by the Model as part of the out-of-core implementation. If sufficient space on the SSD is not available to meet the request, the novfl specification declares that the Model should abort execution rather than overflow to disk. If running CCM3 in an out-of-core fashion (i.e. namelist variables INCORBUF, INCORHST, and INCORRAD are set to .false.), it is important to use the SSD because data transfer rates are on the order of 1000 times faster than to the local disk, and a significant amount of scratch file I/O is done.

Run executable

```
ja
./ccm3bin < nl.init || exit 3
./ccm3bin < nl.restart || exit 3
ja -sclhft
```

Run the Model and gather job accounting statistics. If the code completes successfully (indicated by a return code of zero), great. If not, please see “Model Error Messages” on page 74 or “Trouble-Shooting Model Changes” on page 161.

3. Message Passing Using PVM

This section describes how to run CCM3 under PVM on the SUN, RS6K, and T3D architectures (note, message passing is not required to run on the SUN or RS6K). It is written for the user who is already knowledgeable about PVM operation and terminology. It is also assumed that PVM (version 3.3.10 or later) is built on the host machines and all the proper PVM environment variables ($PVM_ROOT and $PVM_ARCH) are set correctly. Detailed information on how to configure, build, and run PVM is freely available through netlib.

Running CCM3 under PVM on the SUN and RS6K architectures consists of three steps: building the CCM3 PVM executable, configuring and starting the PVM virtual machine, and starting the CCM3 master task.
Configuring CCM3 for PVM requires setting the SPMD token in the header files `preproc.h` and `params.h`. An affirmative answer to the message passing option of the `setup` script will configure the header files correctly. Once the executable is built it must be placed in a directory where PVM can find it. By default, PVM will search first in `$HOME/PVM3/bin/$PVM_ARCH` and then in `$PVM_ROOT/bin/$PVM_ARCH`. The default search paths can be changed by using the host file option `ep=`. If the `ccm3bin` executable is not on a file system shared between the hosts, you must copy the executable to each host manually.

When a user wishes to run a PVM application, a linked collection of computers known as a virtual machine is first created. The virtual machine is created by starting up the PVM daemon on each host. The user may add each host machine interactively or specify all hosts at once by invoking the PVM daemon with a host file. Each line of the host file contains a host name followed by optional parameters. The PVM virtual machine can then be started from a UNIX prompt on any of the hosts. The syntax to start the master PVM daemon (PVMD) is:

```
$PVM_ROOT/lib/pvmd hostfile &
```

When the `pvmd` starts up, it prints out a line similar to either: 80a95ee4:0a9a or /tmp/aaa026175. The master PVM daemon reads the host file and starts slave PVM daemons on all of the machines mentioned in the hostfile. At this point the virtual machine is configured and is ready to run a PVM executable.

Note: The CCM3 uses PVM group library functions. The group server executable (`pvmgs`) must be in the `pvmd` search path. If you change the default executable path using the host file `ep=` option, make sure the group server directory is still included (i.e. either include `$PVM_ROOT/bin/$PVM_ARCH` in the "`ep=`" specification or make a soft link for `pvmgs` in the directory in which `ccm3bin` resides).

Once the virtual machine has been created, the user typically starts the “master” or “initiating” task by hand from a machine within the host pool. This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to integrate the Model forward in time. Before initiating the master CCM3 task though, the user must specify the number of processors on which to run. If the target architecture is SUN or RS6K this is done via the environment variables `NPES`. `NPES` serves essentially the same purpose as `NCPUS` for a target architecture of CRAY, except that its value must be an even number.

The T3D MPP version of PVM uses the hardware capabilities of the T3D instead of a PVM daemon to handle communications between the proces-
Using CCM3

Sec. II.A.

Therefore the user can simply build and run the CCM3 executable without having to build PVM or worry about PVM daemons.

The CCM3 executable can be configured via the setup script to build for the T3D architecture. Note that the Fortran namelist option is not available for the T3D. The CCM and LSM namelist data are therefore hard coded into two new subroutines nidata.F and lsmndata.F. Unfortunately, this requires rebuilding the executable each time the namelist is changed (e.g. when restarting a run).

Once the executable is built, the user will need to set T3D PVM resource environment variables (e.g. PVM_DATA_BUFFERS, PVM_SM_POOL). There is a script called runit included with the code distribution under the T3D subdirectory which sets these variables in a way which will work for the standard model at T42. The script also runs the executable on the desired number of processors via a command-line argument. For other resolutions and configurations, the user may have to experiment and/or reference the T3D PVM documentation: PVM and HeNCE Programmer’s Manual (Cray, 1994).

Since the T3D uses the IEEE standard as its internal data format, the user will also need to allow reading and writing of the Cray formatted boundary and history datasets via the assign command. These specifications are also included in the runit script mentioned above.

The number of processors to use on the T3D is specified via the command-line argument -npes. An example would be:

```
env FILENV=afile -npes 32 ccm3bin
```

This command will run the Model on 32 CPUs. A similar line is included in runit. Printed output will look essentially the same as a non-PVM run with the addition of a few diagnostic messages.
B. Model Input Variables - Namelist Input

Several input files are required in order to execute CCM3. These files include initial and boundary condition datasets, along with two Fortran namelists. The initial and boundary condition datasets supply the grid-point information necessary to solve the prognostic equations through numerical integration (Kiehl et al., 1996). The contents of these datasets are described in detail in Sec. II.D. The namelist input file uses Fortran namelists to set variables that define the run. The focus of this section is to describe in detail the variables that constitute the namelist input. Specific examples of how to set the namelist variables for several different example runs are shown in the next section.

The first namelist read by CCM3, entitled CCMEXP, contains most of the variables required to control a Model run. The second namelist, LSMEXP, contains the input variables required to run the Land Surface Model (LSM) portion of CCM3. For more details about the LSM, refer to A Land Surface Model (LSM Version 1.0) for Ecological, Hydrological, and Atmospheric Studies: Technical Description and User’s Guide (Bonan, 1996). Both of these namelists can be found in the files nl.init and nl.restart (distributed as part of the ccm3.code.tar.gz file).

The namelist CCMEXP is used to set several important variables that control how the Model is run. The information conveyed by the variables contained in CCMEXP can be broken into the following eight categories:

1. Define the type and length of run
2. Input datasets
3. Mass store controls
4. History file options
5. Restart/Regeneration file options
6. Model time
7. Physics controls
8. Memory controls

Table II.1, starting on page 20, displays in detail each of the variables contained in the CCM3 namelist, CCMEXP. The variables are grouped together in the table according to each category listed above, and variable names displayed in bold are required input run the Model (the default values are not adequate or do not exist). Included in the table are the following pieces of information:

- The variable name.
- The 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).
Using CCM3

- The default value of the variable, as set in subroutines preset.F and data.F, or subprogram blkdat.F.

- The column entitled "See Ex." lists the numbers of the relevant examples shown in Sec. II.C, in which the corresponding variable is set.

- The "Description" column is used to describe the purpose of each variable and provide any other relevant details.

Though most of the variables in the CCMEXP namelist are single-valued, some of the variables shown under the "History file options" category in Table II.1 are array-valued. In particular, the variables NDENS, NHTFRQ, MFILT, and NINAVG are each arrays of rank one with an extent defined by ptapes; EXCLUDE and PRIMARY are arrays of rank one with an extent pflds; and AUXF is an array of rank one and extent pflds*(ptapes-1). The parameter ptapes indicates the maximum possible number of history files (primary plus auxiliary) that may declared during a run, and was set equal to six at the time of release of the CCM3 (thus, one primary and a maximum of five auxiliary history files may be output on a single timestep during a run). The parameter pflds specifies the maximum number of fields that can be included in a history file. Both ptapes and pflds are defined in header file pagrid.h. CCMEXP is read into the Model in routine data.F.

**Table II.1: CCMEXP namelist Input Variables**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CASEID</strong></td>
<td>C*8</td>
<td>blanks</td>
<td>1</td>
<td>Case ID is the case identifier. It is saved in the history file header record (refer to Table III.1 on page 123), and is used as part of the MSS path name (see page 67).</td>
</tr>
<tr>
<td><strong>CTITLE</strong></td>
<td>C*80</td>
<td>blanks</td>
<td>1</td>
<td>Case title for header in printed output and history file header.</td>
</tr>
</tbody>
</table>
| **NSREST**    | I    | 0             | 1,3,4,5 | Run type:  
0=initial  
1=restart  
2=regeneration  
3=branch  
| **NESTEP**    | I    | -99999999999 | 1       | Ending timestep (positive) or ending day (negative) of run.† |
| **NELAPSE**   | I    | -99999999999 | 3       | Elapsed time in iterations (positive) or days (negative) to run. May be input instead of NESTEP.‡ |

†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.
### Table II.1: CCMEXP namelist Input Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input datasets</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCDATA</td>
<td>C*80</td>
<td>blanks</td>
<td>1,2, 7</td>
<td>Path and filename of initial dataset. Required for an initial run (NSREST=0).</td>
</tr>
<tr>
<td>BNDTI</td>
<td>C*80</td>
<td>blanks</td>
<td>1,2</td>
<td>Path and filename of time-invariant boundary dataset.</td>
</tr>
<tr>
<td>BNDTYS</td>
<td>C*80</td>
<td>blanks</td>
<td>1,2, 7</td>
<td>Path and filename of time-variant, sea-surface temperatures boundary dataset.</td>
</tr>
<tr>
<td>SSTCYC</td>
<td>L</td>
<td>.TRUE.</td>
<td>7</td>
<td>Flag for yearly cycling SST dataset. If .FALSE., assume multyear dataset.</td>
</tr>
<tr>
<td>BNDTVO</td>
<td>C*80</td>
<td>blanks</td>
<td>1,2</td>
<td>Path and filename of time-variant, ozone mixing ratios boundary dataset.</td>
</tr>
<tr>
<td>OZNCYC</td>
<td>L</td>
<td>.TRUE.</td>
<td>7</td>
<td>Flag for yearly cycling ozone dataset. If .FALSE., assume multyear dataset.</td>
</tr>
<tr>
<td>SETTRACE</td>
<td>C*4</td>
<td>'INIT'</td>
<td></td>
<td>Initialize constituent tracers (only relevant when pcnst &gt; 1): 'INIT' = read from initial dataset NCDATA 'ZERO' = set all to 0.0</td>
</tr>
<tr>
<td><strong>Mass Store controls</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSWRPS</td>
<td>C*8</td>
<td>blanks</td>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>IRT</td>
<td>I</td>
<td>99999</td>
<td>1,2, 6, 9</td>
<td>Retention period in days for history files stored on Mass Store tapes. If NSMVN = 'CTPUBLIC', defaults to 365 (Model imposed maximum is 1825). If a private virtual volume is specified, default is 4095 (Model imposed maximum is 4095). Note, if IRT=0, then no history files will be disposed to the Mass Store (they will be kept as local disk files).</td>
</tr>
<tr>
<td>RIRT</td>
<td>I</td>
<td>99999</td>
<td>6</td>
<td>Retention period for Mass Store restart/regeneration tapes. Same defaults and maximums as IRT. Note, if RIRT=0, then no regeneration files will be disposed to the Mass Store (they will be kept as local disk files).</td>
</tr>
<tr>
<td>NSMVN</td>
<td>C*8</td>
<td>'CTPUBLIC'</td>
<td></td>
<td>Virtual volume for history tape output.</td>
</tr>
<tr>
<td>NRMVN</td>
<td>C*8</td>
<td>'CTPUBLIC'</td>
<td></td>
<td>Virtual volume for regeneration tape output.</td>
</tr>
<tr>
<td>LCROOT</td>
<td>C*40</td>
<td>'/ccm'</td>
<td></td>
<td>Prepend to Mass Store paths for local disk copy (except LSM datasets).</td>
</tr>
</tbody>
</table>
**Table II.1: CCMEXP namelist Input Variables**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>History file options</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STFNUM</td>
<td>I</td>
<td>1</td>
<td></td>
<td>Start history file and regeneration file naming with this number (min=1; max=9999)</td>
</tr>
<tr>
<td>NDENS (ptapes)</td>
<td>I</td>
<td>1,1,1,...</td>
<td>1,5, 9</td>
<td>Array specifying packing densities (1,2 or 4) for primary and auxiliary history files. NDENS&gt;1 is only valid on an NCAR Cray. (see CRAYLIB documentation)</td>
</tr>
<tr>
<td>NHTFRQ (ptapes)</td>
<td>I</td>
<td>-24,0,0,...</td>
<td>1,5, 9</td>
<td>Array of time sample write frequencies for primary and auxiliary history files. If NHTFRQ(i)=0, then no file will be written. If NHTFRQ(i)&gt;0, then frequency is input as number of timesteps. If NHTFRQ(i)&lt;0, then frequency is input as number of hours.</td>
</tr>
<tr>
<td>MFILT. (ptapes)</td>
<td>I</td>
<td>5,5,5,...</td>
<td>1,5, 9</td>
<td>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 disk file or Mass Store tape).</td>
</tr>
<tr>
<td>NLFILT</td>
<td>L</td>
<td>.FALSE.</td>
<td></td>
<td>Flag to put one extra time sample on the first history file of current case. This aids in bookkeeping of long runs.</td>
</tr>
<tr>
<td>NINAVG (ptapes)</td>
<td>C*1</td>
<td>'A','A'...</td>
<td>5,6, 8</td>
<td>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.</td>
</tr>
<tr>
<td>EXCLUDE (pfields)</td>
<td>C*8</td>
<td>blanks</td>
<td>5</td>
<td>List of fields to exclude from the primary history file (these fields would otherwise appear by default on the primary history file; see Sec. II.E)</td>
</tr>
<tr>
<td>PRIMARY (pfields)</td>
<td>C*8</td>
<td>blanks</td>
<td>6</td>
<td>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. II.E).</td>
</tr>
</tbody>
</table>
### Table II.1: CCMEXP namelist Input Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUXF (pflds* (ptapes-1))</td>
<td>C*8</td>
<td>blanks</td>
<td>5,8</td>
<td>Input array used to define the fields to be output on up to 5 auxiliary history files; see Sec. II.E.</td>
</tr>
<tr>
<td>CPUCHEK</td>
<td>L</td>
<td>.FALSE.</td>
<td></td>
<td>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 the Model.</td>
</tr>
<tr>
<td><strong>Restart/Regeneration file options</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSVSN</td>
<td>C*8</td>
<td>blanks</td>
<td>1,9</td>
<td>Filename of restart file. This file contains the name of the master regeneration file to restart from. If NSVSN is not input, no restart or regeneration files will be written. Must be input for a restart run (NSREST=1), not used if a regeneration or branch run (NSREST=2 or 3).</td>
</tr>
<tr>
<td>NREVSN</td>
<td>C*72</td>
<td>blanks</td>
<td>4,5</td>
<td>Filename (or path and filename) of master regeneration file from which to regenerate or branch. Must be an absolute path for a branch run (i.e. must begin with a &quot;/&quot;). Only used for regeneration or branch runs (not restart).</td>
</tr>
<tr>
<td>NREFRQ</td>
<td>I</td>
<td>1</td>
<td>3</td>
<td>Frequency in terms of full primary history files to write restart data (one full primary history file contains MFILT (1) time samples).</td>
</tr>
<tr>
<td>MRESFQ</td>
<td>I</td>
<td>999</td>
<td>8</td>
<td>If primary history file is monthly averaged (NINANG (1) =Q), then MRESFQ is the number of Model days between writing restart/regen datasets (mid-month). Default of 999 implies regeneration files are written monthly.</td>
</tr>
<tr>
<td>MXSZRG</td>
<td>I</td>
<td>380</td>
<td></td>
<td>Maximum size allowed for regeneration dataset (equals either the maximum disk file size or the maximum Mass Store tape size). If MXSZRG&lt; 10^6 units are megabytes, otherwise in bytes.</td>
</tr>
<tr>
<td><strong>Model time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NNDBAS</td>
<td>I</td>
<td>-9999</td>
<td>1</td>
<td>Base Model day for run. If not input, set to ND CUR from initial dataset header.</td>
</tr>
<tr>
<td>NNSBAS</td>
<td>I</td>
<td>-9999</td>
<td>1</td>
<td>Seconds of base Model day for run. If not input, set to NS CUR from initial dataset header.</td>
</tr>
</tbody>
</table>
### Table II.1: CCMEXP namelist Input Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNBDAT</td>
<td>I</td>
<td>-9999</td>
<td>1,7</td>
<td>Base date for run as yymmdd. If not input, set to NCDATE from initial dataset header.</td>
</tr>
<tr>
<td>NNBSEC</td>
<td>I</td>
<td>-9999</td>
<td>1,7</td>
<td>Seconds of base date. If not input, set to NCSEC from initial dataset header.</td>
</tr>
<tr>
<td>DTIME</td>
<td>R</td>
<td>1200.</td>
<td>1</td>
<td>Length of Model timestep in seconds. CAUTION: Changing this variable directly impacts the physical parameterizations in the Model and may impact the Model climate. Changing resolution usually requires a change in DTIME.</td>
</tr>
<tr>
<td>EPS</td>
<td>R</td>
<td>0.06</td>
<td></td>
<td>Time filter coefficient.</td>
</tr>
<tr>
<td>DIF2</td>
<td>R</td>
<td>2.5E5</td>
<td></td>
<td>$V^2$ horizontal diffusion coefficient.</td>
</tr>
<tr>
<td>DIF4</td>
<td>R</td>
<td>1.E16</td>
<td></td>
<td>$V^4$ horizontal diffusion coefficient.</td>
</tr>
<tr>
<td>KMXHDC</td>
<td>I</td>
<td>1</td>
<td></td>
<td>Number of levels over which to apply Courant limiter, starting at top of Model.</td>
</tr>
<tr>
<td>NLVDRY</td>
<td>I</td>
<td>3</td>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>IRADSW</td>
<td>I</td>
<td>-1</td>
<td></td>
<td>Frequency of short-wave radiation calculation in timesteps (if positive) or Model hours (if negative).</td>
</tr>
<tr>
<td>IRADLW</td>
<td>I</td>
<td>-1</td>
<td></td>
<td>Frequency of long-wave radiation calculation in timesteps (if positive) or Model hours (if negative).</td>
</tr>
<tr>
<td>IRADEAE</td>
<td>I</td>
<td>-12</td>
<td></td>
<td>Frequency of absorptivity/emissivity calculations in iterations (if positive) or Model hours (if negative). To avoid having the abs/ems values saved on the regeneration output, this variable should divide evenly with NREFRQ*MFILT(1)*NHTFRQ(1).</td>
</tr>
<tr>
<td>ITSSST</td>
<td>I</td>
<td>1</td>
<td></td>
<td>Frequency of SST update in timesteps.</td>
</tr>
<tr>
<td>DODIAVG</td>
<td>L</td>
<td>.FALSE.</td>
<td></td>
<td>If .TRUE., do diurnal averaging of radiation fields.</td>
</tr>
<tr>
<td>PERTLIM</td>
<td>R</td>
<td>0.0</td>
<td></td>
<td>Maximum size of perturbation to apply to initial temperature field.</td>
</tr>
<tr>
<td>DIVDAMPN</td>
<td>R</td>
<td>0.0</td>
<td></td>
<td>Number of days (from nstep 0) to run divergence damper.</td>
</tr>
</tbody>
</table>

*Appropriate for the T42 resolution*
Using CCM3

Table II.1: CCMEXP namelist Input Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCORRAD</td>
<td>L</td>
<td>.TRUE.</td>
<td>1</td>
<td>If true, then keep abs/ems buffer in-core.</td>
</tr>
<tr>
<td>INCORBUF</td>
<td>L</td>
<td>.TRUE.</td>
<td>1</td>
<td>If true, then keep main Model buffers in-core (buffers b1 and b2).</td>
</tr>
<tr>
<td>INCORHST</td>
<td>L</td>
<td>.TRUE.</td>
<td>1</td>
<td>If true, then keep history buffer in-core.</td>
</tr>
</tbody>
</table>

The namelist LSMEXP is used to set several variables that control how the LSM portion of the Model is run. Table II.2 displays in detail each of the variables contained in the LSMEXP namelist. Unlike the CCMEXP namelist, none of the LSMEXP namelist variables are array valued. LSMEXP is preset and read into the Model in routine lsmini.F. Refer to “Land Surface Model (LSM) Datasets” on page 48 for details describing the input datasets.

Table II.2: LSMEXP namelist Input Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Default Value</th>
<th>See Ex.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDLSF</td>
<td>L</td>
<td>.false.</td>
<td>1,7</td>
<td>Flag to read in an initial dataset.</td>
</tr>
<tr>
<td>FINIDAT</td>
<td>C*80</td>
<td>Default arbitrary initialization</td>
<td>1,7</td>
<td>Initial conditions filename. If not supplied, default is arbitrary initialization (refer to routine lsmtvi.F to see the default values). RDLSF must be set .true. in order for this dataset to be read.</td>
</tr>
<tr>
<td>ESURDAT</td>
<td>C*80</td>
<td>No default - must specify</td>
<td>1,2</td>
<td>Surface data filename. Required input.</td>
</tr>
<tr>
<td>HYDRO</td>
<td>L</td>
<td>.TRUE.</td>
<td></td>
<td>Using prognostic hydrology if true.</td>
</tr>
<tr>
<td>PERGRO</td>
<td>L</td>
<td>.FALSE.</td>
<td></td>
<td>Flag for random perturbation test.</td>
</tr>
<tr>
<td>CONCHK</td>
<td>L</td>
<td>.FALSE.</td>
<td></td>
<td>True for error checks on energy and water conservation.</td>
</tr>
</tbody>
</table>

The Model is capable of executing using default values for most of the namelist variables (most default values are only appropriate at the T42 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, BNDTI, BNDTVS, and BNDTVO. FSURDAT is the only LSMEXP variable that must be set.

Namelist variables that must be explicitly set

Model Input Variables - Namelist Input
C. Initialization and Continuation Runs

The CCM3 offers a single procedure for starting up a Model integration from an initial state, referred to as an “initial” (or “initialization”) run. In addition to an initial run, there are three different options for continuing a Model integration which has halted: a “restart” run, a “regeneration” run, and a “branch” run. These three types of runs are referred to generally as “continuation” runs.

1. Definitions.

An “initial” run (namelist variable NSREST=0) starts the Model from the initial state described by the variables and datasets supplied via the namelist. As the Model integrates forward in time, history datasets and restart/regeneration datasets are written to disk files and possibly the Mass Store. The history datasets describe the state of the atmosphere, and the restart/regeneration files contain the data necessary to perform an exact restart of the Model. The restart file contains the path and filename of the most recently written regeneration files, and the regeneration files contain data from the main Model buffers and common blocks that is necessary to perform a continuation run. Typical initialization runs are shown in Examples 1, 2, 6, 7, 8, and 9, starting on page 29.

CCM3 is often run for very long simulations, requiring CPU time limits which may exceed the mean time between failures, due to environmental problems, system maintenance, etc., on a computer. Therefore, control logic within the Model allows it to be restarted after a case terminates, whether normally or abnormally, and continue running with little or no loss of resources. There are three different kinds of continuation runs:

- **Restart run** (namelist variable NSREST=1) starts from the last timestep at which restart/regeneration output was disposed, and is exact. That is, the results for a restart run will be identical to the results of the case if it had not been interrupted. When performing a restart run, the namelist input should be the same as for the corresponding initialization run with only NSREST and NESTEP (or NELAPSE) changed. Refer to Example 3 on page 34.

- **Regeneration run** (namelist variable NSREST=2) may start from any point in the case at which restart/regeneration output was disposed and is also exact. Regeneration runs are typically used to regenerate one or more history files that may have been lost or corrupted since the case was run. It is required that appropriate regeneration datasets from the same case exist. When performing a regeneration run, the namelist input should be the same as for the corresponding initialization run with only NSREST, NESTEP (or NELAPSE), and NREVSN changed. See Example 4 on page 36.

- **Branch run** (namelist variable NSREST=3) uses the contents of a specified regeneration dataset to start a new case. The history files for this case...
are built "from scratch," and may differ in content, packing density, or any other way from history files of the original case (unlike restart and regeneration runs). For example, the user may conduct a branch run to change the history file(s) to resample subperiods of the original case.

The Model simulation for a branch run will be exactly the same as that of the original initialization run. Therefore, changes in the *namelist* physics controls or timestep are not permitted. Also, because a branch run starts from a "snapshot" of the Model buffers (in the regeneration datasets), changes affecting the length of the main Model buffer, the three-dimensional arrays, etc.—such as adding constituents—are not allowed. Refer to Example 5 on page 39 to see the *namelist* input for a typical branch run.

**2. Example Runs: Setting the Input Variables**

The example input variables in this section illustrate the use of most of the variables in the *namelists* of Sec. II.B. Within the CCMEXP *namelist* input file, *namelist* variables may appear in any order between the delimiters &CCMEXP and &END, and likewise for LSMEXP (see Example 1 below). All *namelist* variables begin in column 2.

The presentation of each of the following examples is the same. An overview of the run is first given, followed by the associated *namelist* input necessary to carry-out the run and a description of certain key variables from the example (these variables are highlighted with a **bold** font in the *namelist*). Most examples describe standard runs of CCM3 on an NCAR Cray at T42 resolution using spectral dynamics.
(a) Example 1 — Initial Run, One Day

The input variables shown in this example request an "initial" run of CCM3 for a one-day simulation on an NCAR Cray. The simulation will start from an initial dataset of 1 September using the datasets from the tar file ccm3.datasets.T42.COS.tar (these datasets can also be found on the NCAR Cray disk under directory /ccm/CCM3/data). This example is distributed as the file n1.init in the CRAY subdirectory of the Model distribution. Output from this initial run is shown in “CCM3 Printed Output” on page 187. Note the E preceding the &CCMEXP is a Cray-specific extension that causes the namelist input to be echoed to standard output.

```
E&CCMEXP
  CASEID = 'clsst01'
  CTITLE = 'Test run: climatological SST'
  NCDATA = 'SEP1.T42.0596'
  BNDTI = 'tibds.T42.0596'
  BNDTVS = 'STR-sst.T42.0596'
  BNDTVO = 'ozn.T42.0596'
  NSWRPS = 'passwd'
  NSVSN = 'rstrt'
  IRT = 100
  NSREST = 0
  NDENS = 2
  NNBDAT = 000901
  NNBSEC = 0
  NNDBAS = 0
  NNSBAS = 0
  MFILT = 20
  DTIME = 1200.
  NSTEP = 72
  NHTFRQ = 72
  INCORBUF = .false.
  INCORHST = .false.
  INCORRAD = .false.
&END
&LSMEXP
  FSURDAT = 'surfdat.T42.0596'
&END
```

<table>
<thead>
<tr>
<th>CASEID</th>
<th>The case identifier for this run is clsst01. 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/regeneration files. The CASEID must be 8 (or less) characters which are valid UNIX filename characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTITLE</td>
<td>This title will appear in the Model printed output and in the history file header.</td>
</tr>
</tbody>
</table>
The initial conditions dataset is located in file SEPI.T42.0596. The Model can start from a CCM2 or CCM3 initial dataset in history file format, packed or unpacked. See “Model Input Datasets” on page 45.

BNDTI contains the time-invariant standard deviation of the topography in file tibds.T42.0596.

BNDTV contains the time-variant sea surface temperatures (SST's) in file STR-sst.T42.0596.

BNDTVO contains the time-variant ozone data in file ozn.T42.0596. This data may be on a number of levels different than the number of Model levels, is on a pressure grid (unlike the hybrid grid of the Model), and is not longitudinally dependent.

This is the Mass Store write password for all output datasets from this run. This password must remain the same throughout a case, or a continuation run cannot update original history files.

NSVSN specifies the filename of the restart file. This file is a text file containing the path and filenames of the latest regeneration datasets from the run. If this variable is not input, no restart or regeneration data will be written.

The history files from this run will be written to the Mass Store with a retention period of 100 days.

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

In this example, the history files are disposed to Mass Store history tapes and are packed 2 to 1. Packing of data is available only on the Cray and results in a loss of significant digits in the data values. Refer to NCAR CRAYLIB documentation.

These time variables are provided to override the times read from the initial dataset header. In this example, NNBDAT NNSEC sets the base date for this run to "000901", or September 1, year 00. Only the month and day of this date are used by the Model unless one or both of the ozone or SST boundary datasets are multiyear.
Using CCM3

**MFILT**

Setting **MFILT** to 20 means that each complete output history file will contain 20 time samples.

**DTIME**

This specifies the Model timestep for the run in seconds (floating point). **CAUTION:** Changing this variable directly impacts the physical parameterizations in the Model and may impact the Model climate. Changes in resolution will probably require a change in **DTIME**.

**NESTEP**

This run will end at timestep 72. The same ending time could have been specified as **NESTEP=−1** (one day for a run with **DTIME=1200**).

**NHTFRQ**

**NHTFRQ** requests the number of timesteps (if a positive value) or the number of Model hours (if negative) between history file writes. A value of 72 (72 timesteps of 1200 seconds each) indicates one write per day. Timestep 0 data will always be written to the history file.

**INCORBUF**, **INCORHST**, **INCORRAD**

False indicates the large data buffers will be stored out-of-core. The target machine for this run is a Cray Y-MP 8-64 with a 256 Mw SSD. This is a relatively small memory Cray, so this run will take advantage of the SSD device. When running on a machine with large memory and no dedicated external high-speed I/O device (SSD), highest speeds are attained by keeping as much of the Model in-core as possible. This reduces the high overhead due to slow disk I/O. The default is that **INCORBUF**, **INCORHST**, and **INCORRAD** are **true**. (large data buffers are stored in-core).

**FSURDAT**

This is the required LSM surface boundary dataset.

**Initialization and Continuation Runs**
(b) Example 2 — Initial Run, T21 on a SUN or RS-6000

The input variables shown in this example request an “initial” run of CCM3 for a non-Cray computer (Sun or IBM).

```
&ccmexp
CASEID = 'init21'
CTITLE = 'Test run'
NCDATA = 'SEP1.T21.0596'
BNDTI = 'tibds.T21.0596'
BNDTVS = 'STR-sst.T21.0596'
BNDTVO = 'ozn.T21.0596'
NSVSN = 'rstrt'
IRT = 0
DTIME = 1200.
NSREST = 0
NDENS = 1
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
MFILT = 20
NESTEP = 72
NHTRQ = 72
INCORBUF = .true.
INCORHST = .true.
INCORRAD = .true.
&end
&lsmexp
FSURDAT = 'surfdat'
&end
```

The leading E (echo namelist to standard out) flag in Example 1 is not used on non-Cray machines. On an IBM RS-6000, the namelist closing delimiter is a forward slash, /, instead of the &end as shown above.

- **NCDATA**: When running on machines other than one of the NCAR Cray’s, the input and output datasets will be in IEEE format.
- **BNDTI**
- **BNDTVS**
- **BNDTVO**: All input and output datasets will be read from and written to the local directory from which the CCM is being run.

- **IRT**: With no MSS, IRT should be set to zero. Output datasets will be written locally with names of the form h0001.

- **NDENS**: Packing must be disabled on non-Cray machines.

- **FSURDAT**: The T21 version LSM surface boundary dataset must be local to the run directory.

If performing a climate simulation (as opposed to a debugging run) other resolution-dependent namelist variables that should be changed are: IRA-
DLW, IRADSW, IRADAE, DTIME, DIFF2, DIFF4, EPS, MFILT and NHTFRQ.
(c) Example 3 — Restart Run, 30 Days

The following example restarts the run from Example 1 and simulates another 30 days. The user has requested that restart and regeneration datasets be written every other time a history file is disposed, in order to save on I/O requests and storage charges. Since \( \text{NHTFRQ}=72 \), 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 \( \text{NINAVG} \) is used (i.e. \( \text{NINAVG}(1) = \text{‘A’} \)), each variable in the time sample will be averaged over the previous 72 timesteps (except \( \text{ORO} \) and \( \text{PHIS} \) which are always instantaneous). NOTE: in a restart run, aside from changing \( \text{NSREST} \) and \( \text{NESTEP} \) or \( \text{NELAPSE} \), the exact same namelist should be submitted as was used in the initial run.

```plaintext
E&CCMEXP
CASEID = ‘clsst01’
CTITLE = ‘Test run’
NCDATA = ‘SEP1.T42.0596’
BNDTI = ‘tibds.T42.0596’
BNDTVS = ‘STR-sst.T42.0596’
BNDTVO = ‘ozn.T42.0596’
NSRPS = ‘passwd’
NSVSN = ‘rstrt’
IRT = 100
NSREST = 1
NDENS = 2
NNBDAT = 000901
NNBSEC = 0
NNDAS = 0
NNSBAS = 0
MFILT = 20
DTIME = 1200.
NELAPSE = -30
NHTFRQ = 72
NREFRQ = 2
INCORBUF = .false.
INCOHST = .false.
INCORRAD = .false.
&END
&LSMEXP
FSURDAT = ‘surfdat.T42.0596’
&END
```

\( \text{NSREST} \)

\( \text{NSREST}=1 \) indicates a “restart” run, which starts from the most recent regeneration datasets and continues the case.

\( \text{NELAPSE} \)

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., \( \text{NELAPSE}=-30 \) means “run 30 days beyond the end of the previous run.” This could also be specified by setting \( \text{NESTEP}=-31(\text{days}) \) or
Using CCM3

NESTEP=2232 (timesteps), assuming a DTIME of 1200 seconds.

NREFRQ Regeneration data for this run will be written only after two history files have been filled (contain 20 time samples) and disposed. Note, since NHTFRQ=72 and MFILT=20, the first regeneration file would be written on the 2880th timestep (72 x 20 x 2), if the Model were to run that long. However, since the run will end after 2232 timesteps, the next set of restart and regeneration files will be written on the last timestep (restart and regeneration files are always written on the last timestep of a run).
(d) Example 4 — Regeneration Run, Retrieve Lost File

If history file 3 (h0003) from the previously run case was corrupted, it can be regenerated. Here NSREST=2 indicates a regeneration run, and NREVSN specifies the appropriate set of regeneration files— those associated with history and regeneration file 2 (r0002) — to continue the run for enough timesteps to build the replacement history file. This procedure differs from the restart in Example 3, because we use regeneration data stored earlier in the case to start the Model run. See Sec. II.E on page 63 for details about regeneration files.

If the lost history file had been file 2 (h0002), we would have had to rerun from the beginning, since due to the setting of NREFRQ, there are no regeneration data associated with file 1. NOTE: As with a restart run, aside from the changes outlined here, use the exact same namelist as the initial run.

```
&CCMEXP
CASEID = 'clsst01'
CTITLE = 'Test run'
NCDATA = 'SEP1.T42.0596'
BNDTI = 'tibds.T42.0596'
BNDTVS = 'STR-sst.T42.0596'
BNDTVO = 'ozn.T42.0596'
NSWRPS = 'passwd'
NSVSN = 'rstrt'
IRT = 100
NSREST = 2
NREVSN = 'r0002'
NDENS = 2
NNBDAT = 000901
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
MFILT = 20
DTIME = 1200.
NELAPSE = -30
NHTFRQ = 72
&END
&LSMEXP
FSURDAT = 'surfdat.T42.0596'
&END
```

NSREST  NSREST=2 indicates a “regeneration” run, which starts from specific regeneration datasets and continues the case.

NREVSN  This variable specifies the regeneration data needed to build the required history file(s). Because we want to regenerate file 3 (h0003), we start with the regeneration data labeled 2 (r0002), associated with the end of the second set of history and regeneration files. Only the name of the “master regeneration dataset” is required — the Model will determine which other datasets need to be read.
(e) Example 5 — Branch Run

In the following example, the user begins a new case 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. A branch run must always start a new case, so the namelist variable CASEID must be changed. If the user desires to change non-history file related variables, it is recommended that an initialization run be conducted instead of a branch run.

In this example, some fields are excluded from the primary history file, and placed on two newly declared 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.

```
E&CCMEXP
  CASEID = 'branch1'
  CTITLE = 'CCM3 branch run'
  NCDATA = 'SEP1.T42.0596'
  BNDTI = 'tibds.T42.0596'
  BNDTVS = 'STR-sst.T42.0596'
  BNDTVO = 'ozn.T42.0596'
  NSVSN = 'rstrt'
  NSREST = 3
  NSWRFS = 'mypass'
  IRT = 365
  RIRT = 365
  NREVSNS = '/USERNAME/csm/INIT2/ccm3/rest/r0345'
  NHTFRQ = 72, 18, 18
  NDENS = 2, 2, 2
  MFILT = 10, 30, 30
  NNBDAAT = 000901
  NNBSEC = 0
  NNDABS = 0
  NNSBAS = 0
  NELAPSE = -30
  NREFRQ = 2
  NINAVG = 'A', 'X', 'M'
  AUXF = '1', 'PRECL', 'PRECC', 'CLOUD',
        '2', 'PRECL', 'PRECC', 'CLOUD'
  EXCLUDE = 'PRECL', 'PRECC', 'CLOUD'
&END
&LSMEXP
FSURDAT = 'surfdat.T42.0596'
&END
```

CASEID

A branch run must always start a new case.

NSREST

NSREST=3 indicates a "branch" run, which starts from a specific set of regeneration datasets and creates a new case.
NREVSN  This run uses master regeneration file r0345 from a previous run to provide the initial data for starting this case.

NHTFRQ  The 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.
(f) Example 6 — Initial Run, Add Fields to History File

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 II.9 on page 53) to the primary history file, and making all of the values in the file instantaneous. Other optional changes are described below.

```
&CCMEXP
  CASEID = 'fields'
  CTITLE = 'CCM3 standard run'
  NCDATA = 'SEP1.T42.0596'
  BNDTI = 'tibds.T42.0596'
  BNDTVS = 'STR-sst.T42.0596'
  BNDTO = 'ozn.T42.0596'
  NSVSN = 'rstrt'
  NSREST = 0
  NSWRPS = 'mypass'
  IRT = 1825
  RIRT = 365
  NDENS = 2
  NNBDAT = 000901
  NNBSEC = 0
  NNNBAS = 0
  NNSBAS = 0
  MFILT = 10
  NELAPSE = -30
  NHTFRQ = 72
  NREFRQ = 2
  NINAVG = 'I'
  PRIMARY= 'PRECSL', 'PRECSC'
&END
&LSMEXP
  FSURDAT = 'surfdat.T42.0596'
&END
```

IRT

The history files from this run will be written to the Mass Store with a retention period of 5 years (1825 days).

RIRT

Regeneration datasets will be written with a Mass Store retention time of 365 days.

NINAVG

Because NINAVG = 'I', all fields on the history file will be instantaneous values.

PRIMARY

This list adds fields to the primary history file. Note that these fields must be in the Master Field List, and the appropriate calls to subroutine ou t f i d . F must be activated. See "Adding New Variables" on page 154 for more information.
(g) Example 7 — Run with Multiyear SST Dataset

Here, the user is using a multiyear SST (Sea Surface Temperature) dataset which runs from 1978 to 1993. For this run, the base date and time information must be set via variables NNBDAT and NNBSEC to fall within the time frame covered by this dataset. Because SSTCYC=.false., the Model will check the year number in searching for the appropriate SST data.

```plaintext
E&CCMEXP
CASEID = 'amip'
CTITLE = 'multiyear SST (AMIP) dataset'
NCDATA = 'DEC16.T42'
BNDTI = 'tibds.T42.0596'
BNDTV$ = 'Dec78-Sep93.sst.T42'
BNDTV0 = 'ozn.T42.0596'
NSVSN = 'rstrt'
NSREST = 0
NSWRPS = 'mypass'
IRT = 3650
RIRT = 365
NDENS = 2
NNBDAT = 781216
NNBSEC = 0
NNDBAS = 0
NNSBAS = 0
SSTCYC = .false.
MFILT = 10
NELAPSE = -3650
NHTFRQ = 72
NREFRQ = 2
&END
&LSMEXP
RDLSF = .true.
FINIDAT = '/CCM3/data/initdat_CCM3.dec16'
FSURDAT = 'surfdat.T42.0596'
&END
```

NCDATA The user has created an initial dataset from December data to use in testing the multiyear SST dataset.

BNDTV$ 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 These variables set the base date and seconds of the model.
NNBSEC  run to be commensurate with dates on the SST dataset. If not input, the base date from the initial dataset will be used.

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

RDLSF  If RDLSF is .true. the LSM initial dataset will be read from the file specified by the namelist variable FSURDAT rather than being generated by the LSM.

FINIDAT  This is the T42 LSM initial dataset for 16 December.
(h) 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 \texttt{mm-yy}, where \texttt{mm} is the month and \texttt{yy} is the year for which the data is valid. In this case, the first file output for September of year 0 will be \texttt{09-00}. Note, since the primary history file is monthly-averaged, the values of \texttt{NHTFRQ} \textbf{(1)} and \texttt{NREFRQ} are ignored.

<table>
<thead>
<tr>
<th>E&amp;CCMEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CASEID = 'monthly'}</td>
</tr>
<tr>
<td>\texttt{CTITLE = 'Monthly average run'}</td>
</tr>
<tr>
<td>\texttt{NCDATA = 'SEP1.T42.0596'}</td>
</tr>
<tr>
<td>\texttt{BNDTI = 'tibds.T42.0596'}</td>
</tr>
<tr>
<td>\texttt{BNDTVS = 'STR-sst.T42.0596'}</td>
</tr>
<tr>
<td>\texttt{BNDTVO = 'ozn.T42.0596'}</td>
</tr>
<tr>
<td>\texttt{NSVSN = 'rstrt'}</td>
</tr>
<tr>
<td>\texttt{NSREST = 0}</td>
</tr>
<tr>
<td>\texttt{NSWRPS = 'mypass'}</td>
</tr>
<tr>
<td>\texttt{IRT = 365}</td>
</tr>
<tr>
<td>\texttt{RIRT = 365}</td>
</tr>
<tr>
<td>\texttt{NDENS = 1,2}</td>
</tr>
<tr>
<td>\texttt{MFILT = 2,2}</td>
</tr>
<tr>
<td>\texttt{AUXF = '1', 'T', 'U'}</td>
</tr>
<tr>
<td>\texttt{NINAVG = 'Q', 'A'}</td>
</tr>
<tr>
<td>\texttt{MRESFQ = 10}</td>
</tr>
<tr>
<td>\texttt{NNBDAT = 000901}</td>
</tr>
<tr>
<td>\texttt{NNBSEC = 0}</td>
</tr>
<tr>
<td>\texttt{NNDBAS = 0}</td>
</tr>
<tr>
<td>\texttt{NNSBAS = 0}</td>
</tr>
<tr>
<td>\texttt{NELAPSE= -61}</td>
</tr>
<tr>
<td>\texttt{NHTFRQ = 72,72}</td>
</tr>
<tr>
<td>\texttt{NREFRQ = 2}</td>
</tr>
<tr>
<td>\texttt{&amp;END}</td>
</tr>
<tr>
<td>\texttt{&amp;LSMEXP}</td>
</tr>
<tr>
<td>\texttt{FSURDAT = 'surfdat.T42.0596'}</td>
</tr>
<tr>
<td>\texttt{&amp;END}</td>
</tr>
</tbody>
</table>

\textbf{AUXF} This run defines one auxiliary history file.

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

\textbf{MRESFQ} This variable indicates that a set of regeneration datasets will be written every 10 days during each month in addition to being written at the end of each month. This prevents the potential loss of up to a full month of the run if a restart becomes necessary.
(i) Example 9 — Debugging Run

This example illustrates settings appropriate for a short 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. For efficiency, no restart/regeneration data will be written. Note the default value of NSREST=0 is used (this implies an initialization run).

```
&CCMEXP
  CASEID = 'debug'
  CTITLE = 'Debugging run'
  NCDATA = 'SEP1.T42.0596'
  BNDTI = 'tibd.T42.0596'
  BNDTYS = 'STR-sst.T42.0596'
  BNDTVO = 'ozn.T42.0596'
  IRT = 0
  NDENS = 1
  MFILT = 10
  NNBDAT = 000901
  NNBSEC = 0
  NNDBAS = 0
  NNSBAS = 0
  NELAPSE = 10
  NHTFRQ = 1
&END

&LSMEXP
  FSURDAT = 'surfdat.T42.0596'
&END
```

<table>
<thead>
<tr>
<th>NSVSN</th>
<th>NSVSN is not set, so no restart/regeneration data are written.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRT</td>
<td>When IRT=0, the output history file is only written to the local disk and not disposed to the Mass Store.</td>
</tr>
<tr>
<td>NDENS</td>
<td>The data are unpacked to maintain full accuracy.</td>
</tr>
<tr>
<td>MFILT</td>
<td>10 time-samples of unpacked data will be written to one disk history file.</td>
</tr>
<tr>
<td>NHTFRQ</td>
<td>History data are written at every timestep.</td>
</tr>
</tbody>
</table>
D. Model Input Datasets

CCM3 requires several input datasets in order to execute. These include an initial dataset containing initial values of Model-predicted variables, and boundary condition and prescribed-value datasets. To allow the use of the CCM Processor in plotting, merging, or otherwise processing the data, these datasets are all in CCM2 history file format (the format of all CCM3 history files is exactly the same as CCM2 history file format). For more information on the Processor, see Introduction to the UNICOS CCM Processor (Buja, 1992).

Boundary and Initial datasets for the Land Surface Model (LSM) contain data over land only. Accordingly, the LSM input datasets are specified via a separate namelist input file. LSM datasets are not in CCM2 history file format, rather they are ASCII text files (Bonan, 1996).

Initial, prescribed-value, and boundary condition datasets for four different resolutions (T42, T31, T21, and T5) are provided (see “How to Build and Run CCM3” on page 9 for information describing how to obtain these datasets). At NCAR, the T42 datasets are all kept on disk under the /ccm/CCM3/data/ directory. For information concerning how/where the Model looks for input datasets, see “CCM3 Disk and Mass Store File Management” on page 145.

1. Initial Dataset

The initial dataset, specified by namelist input variable NCDATA, is attached to the Model job on the initial run of a case, by calling attach.F from subroutine initial.F. On this dataset are initial values of prognostic variables, the surface geopotential field (PHIS) and the orography flag (ORO). The ORO flag values should be consistent with PHIS, and also with the surface type mask which was used in deriving the sea-surface temperatures, and time-invariant surface fields on the other boundary datasets. During the run, ORO will change only in response to changes in sea ice, which for the standard CCM3 is specified in the time-variant SST boundary dataset. All specified boundary data are input to the Model from datasets distinct from the initial dataset and are documented in “Boundary and Prescribed-Value Datasets” on page 46.

The initial dataset is a binary file in the form of a history file, with a three-record header followed by plat latitude data records. See “Model Output Datasets” on page 51 and “History Files” on page 121 for more details about the format of a history file. The initial dataset must contain data for the fields shown in Table II.7.

The CCM Core Group maintains standard initial data files for various resolutions. The T42 datasets reside on permanent disk on each of the NCAR Cray’s in the /ccm/CCM3/data/ directory. Datasets can also be.
Table II.3: Initial Dataset Fields

<table>
<thead>
<tr>
<th>History file Field Name</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PLEV Level Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>temperature (K)</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>zonal wind component (m s(^{-1}))</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>meridional wind component (m s(^{-1}))</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>water vapor specific humidity (kgH(_2)O) / (kg air)</td>
</tr>
<tr>
<td><strong>Single Level Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHIS</td>
<td>(\phi_s)</td>
<td>surface geopotential (m(^2) s(^{-2}))</td>
</tr>
<tr>
<td>PS</td>
<td>(p_s)</td>
<td>surface pressure (Pa)</td>
</tr>
</tbody>
</table>
| 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) |

obtained from the CCM3 tar file, as indicated in “How to Build and Run CCM3” on page 9.

The CCM Processor can create initial datasets by merging fields from one or more history files and building a “save history file.” Processor input control parameters (ICPs) for making an initial dataset appear in Example 2, initial.sh, of the Processor document (Buja, 1992).

2. Boundary and Prescribed-Value Datasets

With the exception of the LSM boundary datasets, all CCM3 boundary and prescribed-value datasets are in history file format. The single time-invariant field required as a boundary condition by a standard run of the CCM3 is on a single dataset containing one time sample. The time-variant fields (monthly values) could theoretically be on a single dataset with multiple time samples, except that the ozone data are unique in two ways: they are defined on vertical levels that differ from those of CCM3, and the data are not longitudinally varying. Therefore, the SST's and the ozone mixing ratios reside on two separate datasets. All time-varying datasets are based on 365 days per year with no leap years. Because these fields are not stored on the regeneration dataset, boundary and prescribed value datasets must be read on a continuation run.
By formatting boundary datasets as CCM2 history files, we provide a standard for these and future datasets and allow the Model to use standard “building blocks,” such as subroutines rdhdr.F and mkslc.F, to read data from these files. In addition, the user may use the CCM Processor to aid in building new boundary datasets to support his/her own research, and analyze these datasets using the CCM Processor.

(a) Time-Invariant Boundary Dataset

The time-invariant topography standard deviation field is on this dataset. The Mass Store pathname is specified as namelist input variable BNDTI. The dataset is read by subroutine tiread.F, called from subroutine intbnd.F, called from the main program ccm3.F. Table II.4 shows the field contained on the time-invariant boundary dataset.

<table>
<thead>
<tr>
<th>History file</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGH</td>
<td></td>
<td>Standard deviation of topography, single level field; (m)</td>
</tr>
</tbody>
</table>

(b) Time-Variant Sea-Surface Temperature (SST) Dataset

The standard SST boundary dataset contains 12 months of data, as 12 time samples. The Model can read SST datasets containing any number of months, including multiyear datasets, as long as the times on the dataset bracket those required by the current simulated time. If any two consecutive time samples are read which are more than 45 days apart, the Model assumes the dataset is not properly written and stops.

The path and filename of the SST dataset is specified as namelist input variable BNDTVS. The dataset is initially positioned by subroutine sstini.F, called from subroutine initext.F. This routine reads through the data records until it has in memory the two records (time samples) which bracket the Model start time. When a new time sample of SST data is required within the time-stepping loop, subroutine advance.F calls routine sstint.F to read the next SST data record.

(c) Time-Variant Ozone, Prescribed-Value Dataset

This dataset contains ozone volume mixing ratios for an arbitrary number of vertical levels. These levels are defined by the hybrid arrays contained in the real portion of the dataset header. CCM3 reads these values and then interpolates to the Model vertical levels at each longitude point on the grid.

The ozone dataset is in the form of a history file containing the fields PS and OZONE. PS is a constant 1000 millibar (100,000 Pascal) pressure field and is provided so the user can request a pressure interpolation to produce pressure plots using the CCM Processor. Because the ozone data do not
vary in longitude, each of the 12 monthly time samples contains plat records of a given number of levels by one longitude.

The pathname of the ozone dataset is specified as namelist input variable BNDTVO. The dataset is read initially by subroutine oznini.F, called from subroutine initext.F, and then within the time-stepping loop by subroutine oznint.F, called from advnce.F. Ozone path lengths for use by the radiation code are computed by subroutine radoz2.F.

3. Land Surface Model (LSM) Datasets

The Land Surface Model requires a time-invariant boundary dataset (FSURDAT) and can optionally accept an initial dataset (FINIDAT). The LSM namelist (LSMEXP) is separate from the CCM namelist (CCMEXP). The LSM datasets are not in CCM2 history file format. The LSM data is read as plain text (ASCII) characters, making the LSM data easily transferable across different computer systems. See Bonan (1996) for more details about the LSM. A description of the LSM input datasets follows.

(a) FSURDAT: The required time-invariant LSM boundary dataset.

The FSURDAT namelist variable specifies the path and filename of the time-invariant LSM boundary dataset. The FSURDAT dataset is required, and is read by the LSM subroutine lsmini.F, which is called from the subroutine initext.F. Table II.5 lists the fields on the time-invariant LSM boundary dataset. All fields are single level.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latixy</td>
<td>latitude</td>
</tr>
<tr>
<td>longxy</td>
<td>longitude</td>
</tr>
<tr>
<td>surf2d</td>
<td>surface type</td>
</tr>
<tr>
<td>soic2d</td>
<td>soil color</td>
</tr>
<tr>
<td>sand2d</td>
<td>percent sand</td>
</tr>
<tr>
<td>silt2d</td>
<td>percent silt</td>
</tr>
<tr>
<td>clay2d</td>
<td>percent clay</td>
</tr>
<tr>
<td>pctlak</td>
<td>percent lake</td>
</tr>
<tr>
<td>pctwet</td>
<td>percent wetland</td>
</tr>
</tbody>
</table>
(b) **FINIDAT: The optional LSM initial dataset.**

The FINIDAT dataset can be used to initialize the LSM with data from a previous LSM integration, and is not required. If FINIDAT is not specified, the LSM will initialize land points to arbitrary values (i.e., the water and temperature values are set to values that are not in equilibrium with the atmosphere model; they are simply provided values that permit the calculation to proceed -- see subroutine `lsmtvi.F` for details). The path and filename is specified as namelist input variable FINIDAT. Additional namelist input variable `rdlsf` must be set to true if using an initial dataset. The dataset is read by subroutine `lsmtvi.F`, called from subroutine `lsmini.F`, called from subroutine `initext.F`. Table II.6 below lists the fields on the optional LSM initial dataset. All fields are single level except as indicated. New initial datasets can be generated by writing these variables to an output file in the LSM format.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2osno</td>
<td>Snow water ( (Kg \cdot m^{-1}) )</td>
</tr>
<tr>
<td>h2ocan</td>
<td>Canopy water ( (Kg \cdot m^{-2}) )</td>
</tr>
<tr>
<td>h2osoi</td>
<td>Volumetric soil water content ( (0 \leq h2osoi \leq \text{watsat}) ) ( (\text{mm}^3_{H_2O})/\text{(mm}^3_{\text{soil}}) ) (multi-level)</td>
</tr>
<tr>
<td>tv</td>
<td>Vegetation temperature ( (K) )</td>
</tr>
<tr>
<td>tg</td>
<td>Ground temperature ( (K) )</td>
</tr>
<tr>
<td>tsoi</td>
<td>Soil temperature ( (K) ) (multi-level)</td>
</tr>
</tbody>
</table>

4. **Slab Ocean Model (SOM) Datasets**

Two datasets are required for the Slab Ocean Model (SOM). These two replace the standard initial dataset and time-varying boundary dataset. See Sec. II.G, page 91, for details about running the SOM.

(a) **NCDATA: The SOM initial dataset.**

When running the SOM, the standard CCM3 initial dataset is replaced by a SOM initial dataset. In addition to the standard CCM3 initial fields (`T, U, V, Q, PHI5, PS, ORO, TS1, TS2, TS3, TS4`), the SOM initial dataset requires a sea ice thickness (`SICTHK`), as well as snow cover (`SNOWN`) to initialize the ice component of SOM as shown in Table II.7.

(b) **TVBDS: The time-varying SOM boundary dataset**

The SOM time-varying boundary dataset, shown in Table II.8, consists of 12 months of ocean mixed layer depth, and ocean mixed layer heat flux.
This replaces the standard time-varying boundary dataset which contains

### Table II.7: Slab Ocean Model Initial Dataset Fields

<table>
<thead>
<tr>
<th>History file Field Name</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PLEV Level Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>temperature (K)</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>zonal wind component (m s(^{-1}))</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>meridional wind component (m s(^{-1}))</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>water vapor specific humidity (\frac{KgH_2O}{Kg_{air}})</td>
</tr>
<tr>
<td><strong>Single Level Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHIS</td>
<td>(\phi_s)</td>
<td>surface geopotential (m^2 s^{-2})</td>
</tr>
<tr>
<td>PS</td>
<td>(p_s)</td>
<td>surface pressure (Pa)</td>
</tr>
<tr>
<td>ORO</td>
<td>ORO</td>
<td>surface type flag: (= 0) over ocean; (= 1) over land; (= 2) over sea ice</td>
</tr>
<tr>
<td>TS1, TS2, TS3, TS4</td>
<td>(T_s)</td>
<td>Four subsurface temperature levels (K) (TS1 is surface temperature)</td>
</tr>
<tr>
<td>SICTHK</td>
<td>(h_i)</td>
<td>sea ice thickness (m)</td>
</tr>
<tr>
<td>SNOWH</td>
<td>(h_s)</td>
<td>Snow cover (liquid water equivalent) (m)</td>
</tr>
</tbody>
</table>

12 monthly averages of SST. The mixed layer depths are specified from observations, and never change from one SOM run to the next, while the 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). The 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 II.8: Slab Ocean Model Time-Varying Boundary Dataset Fields

<table>
<thead>
<tr>
<th>History file Field Name</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Level Fields</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QO</td>
<td>Q</td>
<td>Ocean mixed layer heat flux (W \cdot m^{-2}) (12 monthly averages)</td>
</tr>
<tr>
<td>MLD</td>
<td>(h_o)</td>
<td>mixed layer depth (m) (12 monthly averages)</td>
</tr>
</tbody>
</table>
E. Model Output Datasets

During a CCM3 run, three types of output are produced: 1) a printed log of the run, 2) a series of binary "history files" containing the atmospheric gridpoint data generated during the course of a run, and 3) a series of restart/regeneration files necessary to continue the run. Details of the Model printed output are found in "Printout from a CCM3 Run" on page 69, while the history and restart/regeneration files will be discussed here.

1. Model History Files

The history files contain data values written at specified times during a run. Two classes of history files can be output: "primary" or "auxiliary" history files. A primary history file is the history file output by default. As will be discussed shortly, the primary history file contains many data fields by default, and is written automatically during a run. The exact contents of this file may be customized using namelist variables PRIMARY and EXCLUDE. Auxiliary history files are not written by default, but only if requested by the user via namelist variable AUXF. Up to five auxiliary history files can be declared, each holding fields specified by the user.

The local datasets containing the history files may be transferred to the NCAR Mass Storage System at times specified by the user, if running on a computer linked to the MSS. The history files archive a Model run, and may be viewed by various Model postprocessors.

A history file is written as a series of "time samples," each time sample containing grid-point data for a given time, and consisting of three header records followed by field values stored in data records, one record for each latitude line. An in-depth description of the history file format is given in "History Files" on page 121.

The first three records in a time sample comprise the history file header. Using the three-record header structure, the Model writes the integer, character, and real header values as separate records, making the history file header more easily portable to various computer architectures. The header has three functions:

- On the initial dataset and boundary datasets, it provides some of the information required to start a case.
- On the output history files, it provides the information required by postprocessors to read and locate specific data fields.
- On all datasets of history file format, it serves an archival function, storing information such as date/time and job sequence numbers concerning the Mass Store volume on which it is written.
A complete description of the history file header contents is given in “History File Header Records” on page 123.

The data records in a history file contain floating-point data fields. A single data record is written to the history file for each latitude band. If running multitasked, ordering of the latitude records in the history file is indeterminate. Therefore, each latitude data record contains a latitude index. Complete details of the data record format are given in “Latitude Data Records” on page 129.

The namelist variable NINAVG(i) allows the user to specify the appropriate time averaging of the fields going out to each history file. Possible values are ‘A’ - time averaged, ‘I’ - instantaneous, ‘M’ - point minimum, ‘X’ - point maximum, and ‘Q’ - monthly averaged (‘Q’ only applies to the primary history file). The default for each history file is ‘A’. All fields in a history file are of the type indicated by NINAVG(i), except ORO and PHIS which are always instantaneous. NINAVG(i) controls the field type on both primary and auxiliary history files. The timestep upon which a time sample is written to the history file is specified by NHTFRQ(i). As an example, consider the namelist input shown in Example 3 on page 34. Here, NHTFRQ(1) = 72 and NINAVG is not specified and therefore takes on the default value of ‘A’. Thus 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).

A single history file, primary or auxiliary, will contain the output of MFILT(i) time samples. The packing density of fields in each history file is set with namelist variable NDENS(i) (only valid on an NCAR Cray). A complete description of the history file namelist variables refer to Table II.1 and Table II.2 in Sec. II.B.

(a) Primary History File

Table II.9 contains a list of the fields written to the primary history file. This special list is referred to as the “Master Field List” because it contains the names of all fields that can possibly be written to a primary or auxiliary history file without code modification (to add user-defined fields to the Master field list, refer to “Adding New Variables” on page 154). The first column of this table is entitled “Default File”. If a check mark (√) exists in a given row of this column, then the corresponding field shown in the second column (“Field Name”) will be included on the primary history file by default. If no check mark is present, then the field will not be included on the primary history file by default, but may be included via namelist variable PRIMARY (see “Example 6 — Initial Run, Add Fields to History File” on page 39). Furthermore, any of the fields that are included by default may be removed using EXCLUDE (refer to “Example 5 — Branch Run” on page 37). Some of the field names shown in the second column are represented by two upper-case letters followed by two lower-case x’s (e.g.
<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Q (or TRxx)</td>
<td>$q$</td>
<td>tracer (first is always specific humidity)</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>T</td>
<td>$T$</td>
<td>temperature</td>
<td>N</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>U</td>
<td>$u$</td>
<td>zonal wind component</td>
<td>N</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>V</td>
<td>$v$</td>
<td>meridional wind component</td>
<td>N</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>PS</td>
<td>$p_s$</td>
<td>surface pressure</td>
<td>1</td>
<td>$Pa$</td>
</tr>
<tr>
<td>✓</td>
<td>TS</td>
<td>$T$</td>
<td>surface temperature</td>
<td>N</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>TS1</td>
<td>$T_s$</td>
<td>ice temperature (level 1)</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>TS2</td>
<td>$T_2$</td>
<td>ice temperature (level 2)</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>TS3</td>
<td>$T_3$</td>
<td>ice temperature (level 3)</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>TS4</td>
<td>$T_4$</td>
<td>ice temperature (level 4)</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>PHIS</td>
<td>$\phi_s$</td>
<td>surface geopotential</td>
<td>1</td>
<td>$m^2 \cdot s^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>DCCxx</td>
<td>$\frac{\delta q}{\delta t}$</td>
<td>tracer tendency from adjustment physics</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>TAxxx</td>
<td>$\frac{dq}{dt}$</td>
<td>total advection of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>VDxx</td>
<td></td>
<td>vertical diffusion tendency of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>HAXX</td>
<td></td>
<td>horizontal advection of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>VAXX</td>
<td></td>
<td>vertical advection of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>DFxx</td>
<td></td>
<td>SLT fixer of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>TExx</td>
<td></td>
<td>time tendency of tracer</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>SSxx</td>
<td></td>
<td>tracer source/sinks (pcnst-1 values)</td>
<td>N</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>SFxx</td>
<td></td>
<td>tracer surface fluxes (pcnst-1 values)</td>
<td>N</td>
<td>$Kg \cdot m^{-2} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>DUH</td>
<td>$u$</td>
<td>horizontal diffusive heating rate</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
</tbody>
</table>

**Table II.9: CCM3 Master Field List**

- **Model Output Datasets**
- **Sec. II.E.**
- **Page 53**
### Table II.9: CCM3 Master Field List

<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DVH</td>
<td>$v$</td>
<td>horizontal diffusive heating rate</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>DTH</td>
<td>$\tilde{p}_{TS}(T^*)$</td>
<td>$T$ horizontal diffusion</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>ORO</td>
<td>$S_n$</td>
<td>surface type flag: $= 0$ for ocean $= 1$ for land $= 2$ for sea ice</td>
<td>1</td>
<td>flag</td>
</tr>
<tr>
<td>✓</td>
<td>SNOWH</td>
<td>$S_n$</td>
<td>water equivalent snow depth</td>
<td>1</td>
<td>$m$</td>
</tr>
<tr>
<td>✓</td>
<td>TREFHT</td>
<td>$\eta$</td>
<td>Surface dependent reference height temperature</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✓</td>
<td>LWSH</td>
<td>$\eta$</td>
<td>Liquid water scale height</td>
<td>1</td>
<td>$m$</td>
</tr>
<tr>
<td>✓</td>
<td>ETADOT</td>
<td>$\eta$</td>
<td>Vertical motion on half levels</td>
<td>1</td>
<td>$1/s$</td>
</tr>
<tr>
<td>✓</td>
<td>PRECL</td>
<td>$P_{ls}$</td>
<td>large-scale stable precipitation</td>
<td>1</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>PRECC</td>
<td>$P_{cs}$</td>
<td>convective precipitation</td>
<td>1</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>PRECSL</td>
<td>$P_{cs}$</td>
<td>large-scale stable snowfall</td>
<td>1</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>PRECSCL</td>
<td>$P_{cs}$</td>
<td>convective snowfall</td>
<td>1</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>SHFLX</td>
<td>$c_p \bar{\rho}_1 \langle \bar{w} \theta \rangle_o$</td>
<td>surface sensible heat flux</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>LHFLX</td>
<td>$L \bar{\rho}_1 \langle \bar{w} q \rangle_o$</td>
<td>surface latent heat flux</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>QFLX</td>
<td>$\bar{\rho}_1 \langle \bar{w} q \rangle_o$</td>
<td>surface water flux</td>
<td>1</td>
<td>$Kg \cdot m^{-2} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>PBLH</td>
<td>$u^*$</td>
<td>height of planetary boundary layer</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>USTAR</td>
<td>$u^*$</td>
<td>surface friction velocity</td>
<td>1</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>CGH</td>
<td></td>
<td>pbl nonlocal transport, heat</td>
<td>N</td>
<td>$K \cdot m^{-1}$</td>
</tr>
<tr>
<td></td>
<td>CGQ</td>
<td></td>
<td>pbl nonlocal transport, humidity</td>
<td>N</td>
<td>$1 \cdot m^{-1}$</td>
</tr>
</tbody>
</table>
Table II.9: CCM3 Master Field List

<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>CGS</td>
<td></td>
<td>Counter-gradient coefficient on surface kinematic fluxes</td>
<td>N</td>
<td>$s \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>TPERT</td>
<td>$\theta_o^p$</td>
<td>pbl plume temperature perturbation</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>✔</td>
<td>QPERT</td>
<td>$q_o^p$</td>
<td>pbl plume moisture perturbation</td>
<td>1</td>
<td>$Kg \cdot Kg^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>ZMDQ</td>
<td></td>
<td>Q tendency - Zhang moist convection</td>
<td>1</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>ZMDT</td>
<td></td>
<td>T tendency - Zhang moist convection</td>
<td>1</td>
<td>$Kg \cdot Kg_{air}^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>KVH</td>
<td></td>
<td>diffusivity for heat</td>
<td>N</td>
<td>$m^2 \cdot s^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>KVM</td>
<td></td>
<td>diffusivity for momentum</td>
<td>N</td>
<td>$m^2 \cdot s^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>DUV</td>
<td></td>
<td>$\mu$ vertical diffusion</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>DVV</td>
<td></td>
<td>$\nu$ vertical diffusion</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>DTV</td>
<td></td>
<td>$T$ vertical diffusion tendency</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
<tr>
<td>✔</td>
<td>FSNS</td>
<td>$F_N^S(P_s)$</td>
<td>net downward solar flux at surface</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>FLNS</td>
<td>$F_N^L(P_s)$</td>
<td>net upward longwave flux at surface</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>FLNT</td>
<td>$F_N^S(P_T)$</td>
<td>net upward longwave flux at top of Model</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>FSDS</td>
<td></td>
<td>Flux Shortwave Downwelling at Surface</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>FSNT</td>
<td></td>
<td>net downward solar flux at top of Model</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>CLOUD</td>
<td>$A_c$</td>
<td>cloud fraction</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td>✔</td>
<td>SETLWP</td>
<td></td>
<td>prescribed liquid water path</td>
<td>N</td>
<td>$g \cdot m^{-2}$</td>
</tr>
<tr>
<td>✔</td>
<td>CLDLWP</td>
<td></td>
<td>cloud weighted liquid water path</td>
<td>N</td>
<td>$g \cdot m^{-2}$</td>
</tr>
</tbody>
</table>
Table II.9: CCM3 Master Field List

<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>EFFCLD</td>
<td>(e_A^c)</td>
<td>effective cloud fraction</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>FLNTC</td>
<td>(F_N^L(P_T)_{clr})</td>
<td>net clearsky upward long-wave flux at top</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>FSNTC</td>
<td>(F_N^S(P_T)_{clr})</td>
<td>net clearsky downward solar flux at top</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>FLNSC</td>
<td>(F_N^L(P_S)_{clr})</td>
<td>net clearsky upward long-wave flux at surface</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>FSNSC</td>
<td>(F_N^S(P_S)_{clr})</td>
<td>net clearsky downward solar flux at surface</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>OMEGA</td>
<td>(\omega)</td>
<td>vertical pressure velocity</td>
<td>N</td>
<td>(Pa \cdot s^{-1})</td>
</tr>
<tr>
<td>✓</td>
<td>DQP</td>
<td>(R_{cs} + R_{ls})</td>
<td>(q) tendency from rainout</td>
<td>N</td>
<td>(Kg \cdot Kg^{-1} \cdot s^{-1})</td>
</tr>
<tr>
<td>✓</td>
<td>TAUX</td>
<td>(\tau_\lambda)</td>
<td>zonal surface stress</td>
<td>1</td>
<td>(N \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>TAUY</td>
<td>(\tau_\mu)</td>
<td>meridional surface stress</td>
<td>1</td>
<td>(N \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>SRFRAD</td>
<td>(F_N^{F_R}(P_s))</td>
<td>radiative flux absorbed at the surface</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>QRS</td>
<td>(Q_s)</td>
<td>solar heating rate</td>
<td>N</td>
<td>(K \cdot s^{-1})</td>
</tr>
<tr>
<td>✓</td>
<td>QRL</td>
<td>(Q_{lw})</td>
<td>longwave heating rate</td>
<td>N</td>
<td>(K \cdot s^{-1})</td>
</tr>
<tr>
<td>✓</td>
<td>CLDTOT</td>
<td>(A_c^T)</td>
<td>random overlap total cloud cover</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>CLDLOW</td>
<td>(A_c^L)</td>
<td>random overlap low cloud cover</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>CLDMED</td>
<td>(A_c^M)</td>
<td>random overlap medium cloud cover</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>CLDHGH</td>
<td>(A_c^H)</td>
<td>random overlap high cloud cover</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>TOTLWP</td>
<td></td>
<td>total liquid water path</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>SOLIN</td>
<td>(S_I)</td>
<td>solar insolation</td>
<td>1</td>
<td>(W \cdot m^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>UTEND</td>
<td>(u)</td>
<td>tendency</td>
<td>N</td>
<td>(m \cdot s^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>VTEND</td>
<td>(v)</td>
<td>tendency</td>
<td>N</td>
<td>(m \cdot s^{-2})</td>
</tr>
<tr>
<td>✓</td>
<td>TTEND</td>
<td>(T)</td>
<td>tendency</td>
<td>N</td>
<td>(K \cdot s^{-1})</td>
</tr>
<tr>
<td>✓</td>
<td>LPSTEN</td>
<td></td>
<td>surface pressure tendency</td>
<td>1</td>
<td>(Pa \cdot s^{-1})</td>
</tr>
</tbody>
</table>
Table II.9: CCM3 Master Field List

<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>UTGW</td>
<td>$\frac{\delta u}{\delta t}_{GW}$</td>
<td>gravity wave drag $u$ tendency</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>VTGW</td>
<td>$\frac{\delta v}{\delta t}_{GW}$</td>
<td>gravity wave drag $v$ tendency</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>TAUGWX</td>
<td>$(\tau_\lambda)_{GW}$</td>
<td>gravity wave drag zonal surface stress</td>
<td>I</td>
<td>$N \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>TAUGWY</td>
<td>$(\tau_\mu)_{GW}$</td>
<td>gravity wave drag meridional surface stress</td>
<td>I</td>
<td>$N \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>DTCOND</td>
<td>$\frac{\delta T}{\delta t}_{A}$</td>
<td>$T$ tendency from adjustment physics</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>CMFDT</td>
<td>$\frac{\delta T}{\delta t}_{CC}$</td>
<td>$T$ tendency from moist convection</td>
<td>N</td>
<td>$K \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>CMFDQ</td>
<td>$\frac{\delta q}{\delta t}_{CC}$</td>
<td>$q$ tendency from moist convection</td>
<td>N</td>
<td>$K_g \cdot K_g^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>CMFDQR</td>
<td></td>
<td>rainout (condensation)</td>
<td>N</td>
<td>$K_g \cdot K_g^{-1} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>CMFMC</td>
<td>$M_c$</td>
<td>total convective mass flux</td>
<td>N</td>
<td>$K_g \cdot m^{-2} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>CMFSL</td>
<td>$F_{s-L}$</td>
<td>convective liquid water static energy flux</td>
<td>N</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td></td>
<td>CMFLQ</td>
<td>$F_{q+1}$</td>
<td>convective total water flux</td>
<td>N</td>
<td>$W \cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>CNVCLD</td>
<td>$A_{cc}$</td>
<td>convective cloud fraction</td>
<td>I</td>
<td>fraction</td>
</tr>
<tr>
<td></td>
<td>VT</td>
<td>$vT$</td>
<td>Meridional heat transport</td>
<td>N</td>
<td>$K \cdot m \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>VZ</td>
<td>$v_z$</td>
<td>Meridional transport of gravitational potential energy</td>
<td>N</td>
<td>$m^3 \cdot s^{-3}$</td>
</tr>
<tr>
<td></td>
<td>VQ</td>
<td>$v_q$</td>
<td>Meridional water transport</td>
<td>N</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>VVPUU</td>
<td>$v^2 + u^2$</td>
<td>Kinetic energy</td>
<td>N</td>
<td>$m^2 \cdot s^{-2}$</td>
</tr>
<tr>
<td></td>
<td>RELHUM</td>
<td>$RH$</td>
<td>Relative humidity</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td></td>
<td>Z3</td>
<td>$z$</td>
<td>Geopotential height (above sea level)</td>
<td>N</td>
<td>$m$</td>
</tr>
</tbody>
</table>
### Table II.9: CCM3 Master Field List

<table>
<thead>
<tr>
<th>Default File</th>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>SICTHK</td>
<td>sea-ice thickness; on default file if using the SOM</td>
<td></td>
<td></td>
<td>1</td>
<td>m</td>
</tr>
<tr>
<td>MQ</td>
<td>$q\Delta P$</td>
<td>Water mass</td>
<td>N</td>
<td>$Kg \cdot m^{-2}$</td>
<td></td>
</tr>
<tr>
<td>PSL</td>
<td>$P_{sl}$</td>
<td>Sea level pressure</td>
<td>1</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>OMEGAUP</td>
<td>$\omega^\uparrow$</td>
<td>Average of omega when vertical motion is up</td>
<td>N</td>
<td>$Pa \cdot s^{-1}$</td>
<td></td>
</tr>
<tr>
<td>NUMOMGUP</td>
<td>$N^\uparrow$</td>
<td>Percentage of time vertical motion is up</td>
<td>N</td>
<td>fraction</td>
<td></td>
</tr>
<tr>
<td>CLOUDUP</td>
<td>$A_c^\uparrow$</td>
<td>Average of cloud when vertical motion is up</td>
<td>N</td>
<td>fraction</td>
<td></td>
</tr>
<tr>
<td>DPSLON</td>
<td>See Table II.10</td>
<td>Longitudinal derivative of surface pressure</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
<td></td>
</tr>
<tr>
<td>DPSLAT</td>
<td>See Table II.10</td>
<td>Latitudinal derivative of surface pressure</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
<td></td>
</tr>
<tr>
<td>✓ BEVAP</td>
<td>$\beta_e$</td>
<td>LSM evaporation soil wetness</td>
<td>1</td>
<td>fraction</td>
<td></td>
</tr>
<tr>
<td>✓ BTRAN</td>
<td>$\beta_t$</td>
<td>LSM transpiration soil wetness</td>
<td>1</td>
<td>fraction</td>
<td></td>
</tr>
<tr>
<td>✓ DMI</td>
<td>$\frac{(\Delta M)}{(\Delta t)}$</td>
<td>LSM dry matter production</td>
<td>1</td>
<td>$\mu g \cdot m^{-2} \cdot s^{-1}$</td>
<td></td>
</tr>
<tr>
<td>✓ FCEV</td>
<td>$E_c \left( \frac{c_e^w}{c_e^w + c_i^w} \right)$</td>
<td>LSM canopy evaporation</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
<td></td>
</tr>
<tr>
<td>✓ FCO2</td>
<td></td>
<td>LSM net co2 flux</td>
<td>1</td>
<td>$\mu mol \cdot m^{-2} \cdot s^{-1}$</td>
<td></td>
</tr>
<tr>
<td>✓ FCTR</td>
<td>$E_c \left( \frac{c_e^w}{c_e^w + c_i^w} \right)$</td>
<td>LSM transpiration</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
<td></td>
</tr>
<tr>
<td>✓ FGEV</td>
<td>$\lambda E_g$</td>
<td>LSM ground evaporation</td>
<td>1</td>
<td>$W \cdot m^{-2}$</td>
<td></td>
</tr>
<tr>
<td>✓ FMICR</td>
<td>$R_s$</td>
<td>LSM microbial respiration</td>
<td>1</td>
<td>$\mu mol \cdot m^{-2} \cdot s^{-1}$</td>
<td></td>
</tr>
<tr>
<td>✓ FPPSN</td>
<td>$A^\sun L^\sun + A^\sha L^\sha$</td>
<td>LSM photosynthesis</td>
<td>1</td>
<td>$\mu mol \cdot m^{-2} \cdot s^{-1}$</td>
<td></td>
</tr>
<tr>
<td>Default File</td>
<td>Field Name</td>
<td>Symbol</td>
<td>Field Description</td>
<td>NL</td>
<td>Units</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>--------</td>
<td>-------------------</td>
<td>----</td>
<td>------------------</td>
</tr>
<tr>
<td>✓</td>
<td>FRM</td>
<td>$R_m$</td>
<td>LSM maintenance respiration</td>
<td>1</td>
<td>$\mu$mol $\cdot m^{-2} \cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>FRMF</td>
<td></td>
<td>LSM foliage maintenance respiration</td>
<td>1</td>
<td>$\mu$mol $\cdot m^{-2} \cdot s^{-1}$</td>
</tr>
<tr>
<td></td>
<td>FSA</td>
<td>$S_v + S_g$</td>
<td>LSM absorbed solar radiation</td>
<td>1</td>
<td>W $\cdot m^{-2}$</td>
</tr>
<tr>
<td></td>
<td>FSR</td>
<td></td>
<td>LSM reflected solar radiation</td>
<td>1</td>
<td>W $\cdot m^{-2}$</td>
</tr>
<tr>
<td></td>
<td>H2OSNO</td>
<td>$w_{sno}$</td>
<td>LSM snow, liquid water equivalent</td>
<td>1</td>
<td>Kg $\cdot m^{-2}$</td>
</tr>
<tr>
<td>✓</td>
<td>NDVI</td>
<td></td>
<td>LSM normalized difference veg. index</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>QDRAI</td>
<td>$q_{drai}$</td>
<td>LSM sub-surface runoff</td>
<td>1</td>
<td>mm $\cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>QINFL</td>
<td>$q_{infl}$</td>
<td>LSM infiltration</td>
<td>1</td>
<td>mm $\cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>QOVER</td>
<td>$q_{over}$</td>
<td>LSM surface runoff</td>
<td>1</td>
<td>mm $\cdot s^{-1}$</td>
</tr>
<tr>
<td>✓</td>
<td>RSW</td>
<td>$\sum \theta_i \Delta z_i / \sum \Delta z_i$</td>
<td>LSM root zone volumetric soil water</td>
<td>1</td>
<td>fraction</td>
</tr>
<tr>
<td>✓</td>
<td>TAH</td>
<td>$T_s$</td>
<td>LSM &quot;surface&quot; temperature, height</td>
<td>1</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>TAM</td>
<td>$T_a$</td>
<td>LSM &quot;aerodynamic&quot; temperature, height</td>
<td>1</td>
<td>K</td>
</tr>
<tr>
<td>✓</td>
<td>TGRD</td>
<td>$T_g$</td>
<td>LSM ground &quot;skin&quot; temperature</td>
<td>1</td>
<td>K</td>
</tr>
<tr>
<td>✓</td>
<td>TSOI3</td>
<td>$T_3$</td>
<td>LSM third soil layer temperature</td>
<td>1</td>
<td>K</td>
</tr>
<tr>
<td>✓</td>
<td>TSOI4</td>
<td>$T_4$</td>
<td>LSM fourth soil layer temperature</td>
<td>1</td>
<td>K</td>
</tr>
<tr>
<td>✓</td>
<td>TVEG</td>
<td>$T_v$</td>
<td>LSM vegetation &quot;skin&quot; temperature</td>
<td>1</td>
<td>K</td>
</tr>
</tbody>
</table>
These fields correspond to trace constituents that may be transported. In the history file, the lower-case ‘xx’ in these field names would be replaced by a two-digit tracer number (refer to “Adding Constituents to the Model: The Semi-Lagrangian Transport” on page 156 for more details). The third column, labelled “Symbol”, links the field name to the mathematical symbol used to represent it in *Description of the NCAR Community Climate Model (CCM3)* (Kiehl et al., 1996) or the LSM Technical Note (Bonan, 1996), whichever is appropriate. The next column provides a brief description of the field. In the column labelled “NL” is a mark indicating the number of levels on which the field has data. A “1” indicates a single-level field, and an “N” indicates a multilevel field (with data present on plev vertical levels). The last column in the table shows the physical units associated with each field.

**Adding non-default fields to a history file**

Within the CCM3 code, fields to be written to the history file must include a call to subroutine `outflfd.F` (see Sec. III.E and Sec. IV.B for more details). Due to efficiency considerations, calls to `outflfd.F` for most “inactive” fields (i.e. those not on the default primary history file) are commented in the code. Therefore, 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 `outflfd.F` call for this field.

In CCM3 the order in which fields are written to the history file is determined at run time. Hence, the order in which the default fields appear in Table II.9 does not reflect the actual order of fields on the primary history file.

**Monthly-averaged primary history file**

CCM3 includes an option 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 only one time slice of data (regardless of the value of MFILT (1)).

In addition to those fields shown as default in the Master Field List, a monthly-averaged primary history file includes an additional 14 fields. These fields are shown again in Table II.10. Because of 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. See “Creating and Running the Executable in Batch Mode” on page 13 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 II.10: Additional Fields on a Monthly-Averaged Primary History File

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Symbol</th>
<th>Field Description</th>
<th>NL</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT</td>
<td>$v_T$</td>
<td>Meridional heat transport</td>
<td>N</td>
<td>$K \cdot m \cdot s^{-1}$</td>
</tr>
<tr>
<td>VZ</td>
<td>$v_z$</td>
<td>Meridional transport of gravitational potential energy</td>
<td>N</td>
<td>$m^2 \cdot s^{-3}$</td>
</tr>
<tr>
<td>VQ</td>
<td>$v_q$</td>
<td>Meridional water transport</td>
<td>N</td>
<td>$m \cdot s^{-1}$</td>
</tr>
<tr>
<td>VVPUU</td>
<td>$v_T^2 + u_z^2$</td>
<td>Kinetic energy</td>
<td>N</td>
<td>$m^2 \cdot s^{-2}$</td>
</tr>
<tr>
<td>RELHUM</td>
<td>$RH$</td>
<td>Relative humidity</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td>Z3</td>
<td>$z$</td>
<td>Geopotential height (above sea level)</td>
<td>N</td>
<td>$m$</td>
</tr>
<tr>
<td>MQ</td>
<td>$q\Delta P$</td>
<td>Water mass</td>
<td>N</td>
<td>$Kg \cdot m^{-2}$</td>
</tr>
<tr>
<td>PSL</td>
<td>$P_s$</td>
<td>Sea level pressure</td>
<td>I</td>
<td>$Pa$</td>
</tr>
<tr>
<td>OMEGAUP</td>
<td>$\omega^{+}$</td>
<td>Average of omega when vertical motion is up</td>
<td>N</td>
<td>$Pa \cdot s^{-1}$</td>
</tr>
<tr>
<td>NUMOMGUP</td>
<td>$N^+$</td>
<td>Percentage of time vertical motion is up</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td>CLOUDUP</td>
<td>$A_c^+$</td>
<td>Average of cloud when vertical motion is up</td>
<td>N</td>
<td>fraction</td>
</tr>
<tr>
<td>DPSLON</td>
<td>$(\frac{\partial}{\partial \mu} \ln p_s)/a$</td>
<td>Longitudinal derivative of surface pressure</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
<tr>
<td>DPSLAT</td>
<td>$(\frac{1 - \mu^2}{\mu} (\frac{\partial}{\partial \mu} \ln p_s))/a$</td>
<td>Latitudinal derivative of surface pressure</td>
<td>N</td>
<td>$m \cdot s^{-2}$</td>
</tr>
</tbody>
</table>

(b) Auxiliary History Files

CCM3 contains a multiple history file capability. 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 the `namelist` variable `AUXF`. The capability for writing auxiliary history files gives the user great flexibility in the fields which are archived, and the frequency at which these data are written. Auxiliary files may contain the same or different fields as the primary history file, and 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 five auxiliary history files.

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 are 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” on page 37.

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 history volumes. An auxiliary file may be added on a continuation run if desired.

(c) Naming the Primary and Auxiliary History Files

A simple naming scheme for the history files has been implemented. First consider a primary history file in which the field types are either A, I, M, or X (but not Q). The name of this file will consist of an “h” followed by a four-digit number. For example, on a typical initialization 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 for the primary history file. First, if namelist variable STFNUM is set, then naming of the history files will begin with this number. For instance, if STFNUM=0123, 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 month and year of the Model date. For instance, if the primary history file is being written at the end of the ninth month of year 00, then the file name would be 09-00 (also see “Example 8 — Generate a Monthly-Averaged History File” on page 42).

Naming of the auxiliary history files is very similar to the primary history file. The first auxiliary history 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, ha0003 will be generated for the first auxiliary history file, and hb0001, hb0002, hb0003 for the second auxiliary history file, etc.
2. Restart and Regeneration Datasets

In Sec. II.C, initialization and continuation runs were defined and several examples were given showing how to perform such runs. Recall that there are three types of continuation runs: a restart run, a regeneration run, and a branch run. A restart run is an exact continuation of a previous run from its point of termination. Regeneration runs are typically conducted in order to regenerate corrupt or missing history files. A branch run, unlike a restart or a regeneration run, is a new case that uses the regeneration datasets from a previous run to begin the integration.

Regardless of the type of continuation run being conducted, information is obtained by reading the regeneration files output from a previous run. There are several types of regeneration files written during a run. Since these files are meant only to be read by the Model when performing a continuation run (and not looked at by the user), their contents will not be discussed here. What will be described is the type of regeneration files that are written during a run, how they are named, and what namelist variables affect how frequently they are written. A more detailed discussion of the regeneration files, including a general description of their contents and related coding issues, is presented in “Restart/Regeneration Files” on page 138.

(a) Types of Restart and Regeneration Files

There is one restart file and four different types of regeneration files that may be written during a run. The “restart” file is simply an ASCII text file that holds the path and filenames of the most recently written regeneration files. The four types of “regeneration” files are the master, primary, secondary, and history buffer regeneration files. These files contain grid-point data and other information necessary to continue a run.

The write frequency of the restart/regeneration files depends on how the associated namelist variables are set. Refer to Table II.1 on page 20 for a summary of the related namelist variables, and also to Examples 1 through 4 in Sec. II.C. If NSVSN is set, then restart/regeneration datasets will be written during the run. The frequency at which these files are written is a function of NREFRQ (unless the fields on the primary history file are monthly-averaged, then the write frequency is specified by MRESFQ), NHTFRQ and MFILT. Of the restart and regeneration files, only the restart file and the master and primary regeneration files are certain to be written at the time specified by NREFRQ or MRESFQ. Writing of the secondary and history buffer regeneration files is contingent upon additional aspects of the run.

In the case of secondary regeneration files, they are written only if the interval of absorptivity/emissivity calculations (IRADAE) does not evenly divide into the frequency of writing to the regeneration files (NREFRQ x MFILT(1) x NHTFRQ(1)). If this is the case, then the absorptivity/emis-
sivity grid-point values must be saved since they will not be calculated on the first timestep of a restart or regeneration run.

History buffer regeneration files are written only if a declared history file (either primary or auxiliary) is not written on the same timestep as the restart/regeneration dataset. This may occur during a run in which the primary history file is monthly-averaged, if the run terminates on a timestep in which history files are not written, or if auxiliary history files are being written. One history buffer regeneration file will appear for each declared history file that was not written. For example, if two auxiliary history files are declared along with the primary, and the Model terminates on a timestep in which only the first auxiliary history file was written, then history buffer regeneration files will be written for the primary and second auxiliary history files.

Normally, restart/regeneration datasets are only written on a timestep when the primary history file is written. However, when the primary history file is monthly-averaged, a significant amount of computing time can elapse between the monthly writing of the regeneration files. Thus, the user may wish to write the restart/regeneration files more frequently than once per simulated month. The user may specify that restart/regeneration datasets be written periodically throughout a month when doing monthly averaging, using the namelist variable MRESFQ. These files are referred to as “mid-month” restart/regeneration files. Be sure to read the information on restart datasets about the related namelist variables in Table II.1 before requesting a monthly-averaged primary history file. Mid-month regeneration files are given short day MSS retention times by default unless otherwise requested by the namelist variable RIRT.

Primary, secondary, and history buffer regeneration files contain grid-point information and can become very large. The Model calculates the size of these datasets during its initialization phase, and if the size of any one of these files is larger than MXSZRG (see Table II.1), then the dataset will be split over more than one disk file. At the release of CCM3, MXSZRG defaulted to 380 Mb (the current capacity of a Mass Store tape). Note, at T42 resolution, the primary regeneration file (largest of the regeneration files) is approximately 36 Mb.

(b) Naming of Restart and Regeneration Files

With the understanding that restart/regeneration files are only meant to be read by the Model, and that the information relevant to the user concerning these files is only their names (and directory paths), the remaining discussion will address how these files are named. As indicated in the description of the namelist input (Table II.1) and demonstrated in the namelists of the Example 2 through 4 (Sec. II.C), only the names of the restart and master regeneration files are needed as input for a continuation run. Thus, the method of naming the these files will be discussed here in detail. The
Model is able to determine what other regeneration files (i.e. primary, secondary, etc.) need to be read from data saved on the master regeneration file.

The name of the restart file is specified by namelist variable NSVSN. Only one copy of this file exists, since by definition, it contains the filenames of the most recently written restart and regeneration files. Each time the restart file is written to, the file named via NSVSN is overwritten.

Naming of the master regeneration file is done automatically, and mimics the name of the primary history file. Except for a few special cases, the name of the master regeneration file begins with an "r" and ends with the same number as the primary history file. There are four different types of names that may arise for the master regeneration file, each addressed in the examples below:

1) Consider a case in which the primary history file contains fields of type A, I, M, or X (but not Q, monthly-averaged). Assume that the fourth primary history file has been filled with MFILT(1) time samples and that the restart/regeneration files are due to be disposed with this primary history file. The default filename of the primary history file would be h0004. The corresponding name of the master regeneration file would be r0004.

2) Once again consider a case in which the primary history file contains fields of type A, I, M, or X. Assume the sixth primary history file, h0006, is being written to but is not full when the run terminates (contains less than MFILT(1) time samples). The name of the master regeneration file written when the run terminates will be r0006.aa. Thus, if the primary history file is not full on the final timestep of a run, then the name of the master regeneration file will have a ".aa" suffix (note, all of the regeneration files written previous to the final timestep will be named as in (1) above -- without the ".aa" suffix).

3) Now consider a case in which the primary history file contains monthly-averaged fields (type Q). At the end of each simulated month, the primary history file will be written along with the master regeneration file. As discussed previously, the name of the primary history file will be composed of the month and year of the Model date. A typical example would be 09-00, as described in "Example 8 — Generate a Monthly-Averaged History File" on page 42. The corresponding master regeneration file would be named r09-00. Note, if the user wished to start a continuation run beginning in October, then the master regeneration file labeled with the September month number should be used (namelist variable NREVSN)
4) As a final case, assume the fields of the primary history file are monthly-averaged. The Model is capable of writing “mid-month” restart and regeneration files as directed by namelist variable MRESFQ. In this case, restart and regeneration files are written every MRESFQ days during a month, as well as along with the primary history file at the end of a month. Any master regeneration file written at “mid-month” intervals is named with a two letter suffix. For example, consider a case in which the primary history file to be written at the end of the current month will be named 07-85, and that MRESFQ will cause “mid-month” restart and regeneration files to be written twice during the month. The names of the two master regeneration files that will be written are r07-85.aa and r07-85.ab. If more than two mid-month regeneration files were written, then the suffixes would continue to increase as “.ac”, “.ad”, etc. At the end of this Model month, the master regeneration file would be named as indicated in (3) above: r07-85.

The exact filenames of the primary, secondary, and history buffer regeneration files are less important since the user never has to deal with them directly (the Model generates the filenames automatically for writing or reading). However, they will be mentioned here briefly to alert the user of the many files that may appear and to avoid having them confused with the master regeneration filename.

The primary, secondary, and history buffer regeneration files all have the same root name as the master regeneration file, but with different suffixes appended. The primary regeneration file will have an upper case letter appended, such as “.A” or “.B”. Secondary regeneration files have a lower case letter appended to the root name, such as “.a” or “.b”. The history buffer regeneration file for the primary history file will have a “.h0” appended to the root name. The first auxiliary history file will have a “.ha” appended, the second a “.hb”, etc.

For example, consider case (1) described above, in which the primary history file h0004 was disposed along with master regeneration file r0004. Assume that primary and secondary regeneration files were also written, along with history buffer regeneration files for two auxiliary history files. The complete set of regeneration files that would be written in this instance is listed below:

- r0004 - master regeneration file
- r0004.A - primary regeneration file
- r0004.a - secondary regeneration file
- r0004.ha.1 - history buffer regeneration file for first auxiliary history file
- r0004.hb.1 - history buffer regeneration file for second auxiliary history file
If the primary regeneration file was large enough (\( \textgreater \text{MXSZRG} \)) to be split over two files, then file \( r0004.B \) would exist in addition to \( r0004.A \). Similarly, if the secondary regeneration file was split into two files, then \( r0004.b \) would also be present. If the first auxiliary history buffer regeneration file were to be split across two files, the second of these files would be named \( r0004.ha.2 \). Note, any additional suffixes associated with mid-month regeneration or Model termination (i.e., \(".aa",".ab",\) etc.; see cases (2) and (4) above), would be tacked on to the end of these filenames (i.e. \( r0004.ha.1.aa \), etc.).


The history files, regeneration datasets, and restart dataset can be archived on the NCAR Mass Storage System if running the Model on a computer that has a connection to the MSS. These files are transferred asynchronously to the MSS by the Model as they are completed, if \textit{namelist} variables IRT and RIRT are non-zero. If IRT is zero, then the history files will not be disposed to the MSS, but retained as local disk files. Similarly, if RIRT is zero, then the restart and regeneration files will not be disposed to the MSS, but retained as local disk files.

The Mass Store pathnames for these transfers are automatically generated by the Model. For history files, the pathname is as follows:

\[
\text{//USERNAME/csm/CASEID/ccm3/hist/hxxxx}
\]

Here, \texttt{USERNAME} is the upper-case equivalent of the user’s Cray login name, i.e., the user’s root directory on the Mass Store System, and \texttt{CASEID} is the case identifier and is set via the \textit{namelist} input. For regeneration and restart datasets, the following pathname is used:

\[
\text{//USERNAME/csm/CASEID/ccm3/rest/rxxxx}
\]

The Model imposes a maximum on Mass Store retention times for files written to virtual volume \texttt{CTPUBLIC} of 5 years, or 1825 days, and for those written to private virtual volumes of 4095 days (over 11 years). The NCAR Mass Storage System currently limits the size of a tape to 380 Megabytes. The user must choose values of \textit{namelist} variable \texttt{MFILT(i)} such that individual history files do not grow beyond this limit. The user may further reduce the size of history files by using the packing feature available on the Cray PVP architectures (refer to NDENS in Table II.1 on page 20). The \textit{namelist} variable \texttt{MXSZRG}, is provided to automatically limit the size of regeneration datasets. Regeneration datasets will be split over several disk files (tapes) if necessary to satisfy the maximum file size of \texttt{MXSZRG}. 
It is recommended that the user specify a non-blank write password, NSWRPS. File passwords are the only form of security available on the Mass Storage System.
F. Model Printed Output

This section documents the printed output that results from running CCM3. A complete printout from a successful CCM3 run is shown in "Appendix C: CCM3 Printed Output." Selected portions of this printout are reproduced and explained in “Printout from a CCM3 Run” below. “Model Error Messages” on page 74 lists all CCM3 error messages exactly as they would appear in the printed output, followed by a brief explanation of the message.

1. Printout from a CCM3 Run

The following printed output was produced by the sample namelist in “Example 1 — Initial Run, One Day” on page 29 (also distributed with the CCM3 tar file in file CRAY/n1.init). The code was run on an NCAR Cray connected to the MSS at T42 resolution, with spectral dynamics using the standard ocean/ice model. The entire printout appears in Appendix C.

Your printout may echo certain commands from the run script if executed in batch mode. A segldr statistical summary is requested by this script and will appear at the beginning of the printout when run on an NCAR Cray. The first output from the Model itself declares it as NCAR CCM3 and documents the Model version number and the starting date and time of the run:

Because of the “E” in column one of the namelist group designator CCMEXP, all namelist records are echoed to the output file. This is followed by a summary of namelist variables, with explanations.

Next you will see the hybrid coordinate layer locations and reference pressures are printed from subroutine hycoef.F as the initial datasets are read.

Information from the boundary dataset headers along with a description of the and initialization of the Land Surface Model (LSM) is printed as the datasets are attached and positioned to the appropriate date. The LSM initial printout includes the starting date and time of the run as well as multi-tasking information. For parallel efficiency, the LSM uses a contiguous
array of land points which are split into a number of parallel tasks. In the T42 example shown, the LSM land array is divided into 39 separate vectors (approximately equal to KPT/150, where KPT is the total number of LSM land points including subgrid points.). The variable NUMLV (defined in preproc.h) determines the number of parallel tasks for the LSM. Following this are descriptions of the run control variables and boundary datasets as determined by the LSM namelist. In this example no initial LSM boundary data was specified so a default arbitrary initialization is used. Note that the LSM must use the same grid resolution as CCM3. The parameters which specify the resolution for the LSM are contained in the header file preproc.h.

```
LSM land surface model, version 1

* initial run at:*
  nstep= 0 year= 0 month= 9 day= 1 seconds= 0

time step = 1200. s except for soil hydrology which uses a 600. s time step

* multi-processing: there will be 39 calls to the vector-based code*
  1 calls with vector length  157
  38 calls with vector length  154

* preset fortran unit numbers:*
  unit 5 = standard input
  unit 6 = standard output

* logical run control parameters:*
  use initial data set = F
  prognostic hydrology = T
  flag for random perturbation test = F
  energy and water conservation checks = F

* data files:*
  surface data = surfdat.T42.0596
  initial data = arbitrary initialization

* grid characteristics:*
  latitude points = 64
  maximum longitude points = 128
  minimum longitude points = 128
  land points on grid = 2796
  total land points including subgrid points= 6009

* successful set up of lsmtc common block

* successful set up of lsmtv common block

* successful set up of LSM
```

Model Printed Output
As described in “Data Structures” on page 103, CCM3 has the capability of assigning several large data structures to an external I/O device (out-of-core) or running with these data structures contained in memory (in-core). The in-core and out-of-core data statistics are provided after all these data structures are allocated. The example printout shows that the Model is configured to run with 11Mw of out-of-core data files.

<table>
<thead>
<tr>
<th>Size of out-of-core files:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model buffer = 2228224</td>
</tr>
<tr>
<td>Exbuf = 917504</td>
</tr>
<tr>
<td>History buffer = 4194304</td>
</tr>
<tr>
<td>Abs/ems = 3801088</td>
</tr>
<tr>
<td>Total = 11141120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size of files held in core:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total = 0</td>
</tr>
</tbody>
</table>

A table, “Summary of Logical Unit Assignments,” which lists the logical units used by the Model is printed next. It is important for the user to refer to this table in identifying unit numbers from system error messages.

**** Summary of Logical Unit assignments ****

| Initial dataset unit (ninit) = 4 |
| Time-inv boundary dataset (nbndti) = 1 |
| Ozone dataset unit (nozone) = 2 |
| SST dataset unit (nsst) = 3 |
| Model buffer scratch file (nral) = 11 |
| Model buffer scratch file (nrb1) = 21 |
| Exbuf scratch file (nspilt) = 22 |
| History buffer scratch file (sunit)= 60 |
| Abs/ems scratch file (nabem) = 70 |
| History tape number = 20 |
| Restart dataset unit (nsds) = 7 |
| Master regeneration unit (nrg) = 8 |
| Regeneration dataset units (nrg1) = 9 10 12 13 14 |
| Abs/ems unit for restart (nrg2) = 15 16 17 18 19 |
| Regeneration units for hist tape 1 = 23 24 25 26 27 |

When the Model enters the time-stepping loop, information about the global state is printed:

<table>
<thead>
<tr>
<th>NSTEP</th>
<th>RMSZ</th>
<th>RMSD</th>
<th>RMST</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSTEP = 0</td>
<td>8.852169942333591E-05</td>
<td>6.563139033659272E-06</td>
<td>252.300 ...</td>
</tr>
<tr>
<td>STPS</td>
<td>STQ</td>
<td>HOR</td>
<td>VERT</td>
</tr>
<tr>
<td>9.84579E+04</td>
<td>2.403683313974500E+01</td>
<td>0.93</td>
<td>0.14</td>
</tr>
</tbody>
</table>
where:

RMSZ is the Global RMS Vorticity
RMSD is the Global RMS Divergence
RMST is the Global RMS Temperature
STPS is the Global Mass Integral
STQ is the Global Moisture Integral

COURANT HOR is the Maximum Courant Number for the Horizontal Velocity Field
COURANT VERT is the Maximum Courant Number for the Vertical Velocity Field

The first time a history file header is written for a history file (primary or auxiliary), the entire header is printed as follows:

*** HEADER FOR CCM3 HISTORY FILE ***

*** Primary History File ***

CASE: clsst01
TITLE: Test run: climatological SST

LENHDI MFTYP MFILH MFILTH NRBD MAXSIZ . . .
292 43 1 20 3 33662

PLAT PLEV PTRM PTRN PTRK NFLDH . . .
64 18 42 42 42 85

FIELD LIST

<table>
<thead>
<tr>
<th>FLD NO.</th>
<th>NAME</th>
<th>FLG.</th>
<th>FLD PT.</th>
<th>PACK.</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q</td>
<td>12</td>
<td>3</td>
<td>2</td>
<td>KG/KG</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>12</td>
<td>1191</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>3</td>
<td>U</td>
<td>12</td>
<td>2379</td>
<td>2</td>
<td>M/S</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>83</td>
<td>SNOWH</td>
<td>10</td>
<td>31221</td>
<td>2</td>
<td>M</td>
</tr>
<tr>
<td>84</td>
<td>UTGW</td>
<td>12</td>
<td>31287</td>
<td>2</td>
<td>M/S2</td>
</tr>
<tr>
<td>85</td>
<td>VTGW</td>
<td>12</td>
<td>32475</td>
<td>2</td>
<td>M/S2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The contents of this header is described in detail in "History File Header Records" on page 123. As time-stepping continues, global integrals are printed associated with each value of nstep fields are printed, followed by abbreviated header printouts as additional history files are written. When it is time to transfer output files to the Mass Store, information concerning this operation is printed. Subroutine savdis.F calls system utility mswrite, which issues the command to dispose the file to the MSS. A copy of the command is printed for the user's information. The following printout shows the options used in conjunction with mswrite and, in the case of the history file, gives other information about the output data.
At the end of the run is printed information about the length of the simulation:

```
Number of completed timesteps:  72
Time step  73 partially done to provide convectively adjusted and time filtered values for history tape.

-----------------------------
****** END OF MODEL RUN ******
-----------------------------
```

Job accounting information is summarized by the `ja` command. In the “Job Accounting — Command Report” table below, the entry starting with "ccm3bin" shows the elapsed, user, and system CPU seconds used in running the Model, and the memory “hiwater” mark, which is designated in “clicks,” or 512-word blocks. Thus, this run began at approximately 16:00, and executed for 549 (user) CPU seconds in 386 elapsed (wall clock) seconds. At the point of greatest memory usage, the Model used 20128 “clicks,” or about 10 Megawords of main memory. Note, the job accounting command is only available on a Cray.

```
Job Accounting - Command Report
===============================
Command Started Elapsed User CPU Sys CPU . . Memory
Name    At    Seconds Seconds Seconds . . HiWater
======== ====== ========= ========= ========= . . =======
ccm3bin 16:00:25 386.8616 549.4706 17.8048 . 20128

A detailed breakdown of actual GAU (general accounting units) charges will follow this summary.

2. Model Error Messages

All errors trapped by the Model 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 for the user is included.

**ATCHBND**:Failed to open file xxxxxx ierr=nn

The requested boundary dataset file was found on the Cray disk as xxxxxx, but the `open` statement failed.
ATCHBND: Local dataset name xxxxxx exceeds 64 characters.

The path name for the requested boundary dataset must be less than 64 characters in length.

ATTACH: Error from msread -- Message from merror

ATTACH: Local file = /xxx/xxxxxx
Remote pathname = /xxx/xxx/xxxxxx

If file /xxx/xxxxxx is a midmonth restart, it may not have been put on Mass Store yet, or it may have already aged off. If a midmonth file has already aged off, the model may be regenerated from the previous months restart files.

The msread (Mass Store read) command in ATTACH returned the indicated error code.

ATTACH: Failed to open file xxxxxx ierr=nn

The requested file was found on the Cray disk, but the open statement failed.

BLDFLD: Too many fields -- pflds, if=nnn mmm

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

A history file number was requested which exceeded 9999. Formatting of history file names is limited to “h” plus 4 digits (see “Naming the Primary and Auxiliary History Files” on page 62).

CALDYI (or CALENDR): Invalid base day of base date input: ddd

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

CALDYI (or 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.
CALDYI (or CALENDR): Invalid base seconds(mbsec):  

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

CALDYI (or CALENDR): Invalid base seconds(msbase):  

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

CCM3: PVM error-Gave up waiting for drv to enroll
CCM3: PVM error - fstatus info = nn
CCM3: PVM error - atm waited ss wall clock seconds

CSM driver failed to enroll within 300 seconds. Only valid in coupled Model simulation.

CMPLR: Entered with incorrect dimension

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 nn iterations
Eigenvalue=nn

The algorithm that computes the Gaussian weights has failed to converge in nn 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 nnn ***

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,pgl= 11 mm nn

IF on T3D then increase number of latitude bands on processor (increase parameter numbnd) otherwise this is an early indication that the Model is in the process of blowing up.

DADADJ: Divergence in dry adiabatic adjustment
lat,lon = nn,nn zeps= nn

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= nn nn

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 caseid 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: 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: 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
See Table II.1 on page 20.

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:***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: Pathname for regeneration file too long.
NREVSN was input with more than 80 characters.

DATA: Starting tape 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: Too many auxiliary tapes 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

FACTOR: Problem size is not divisible by 2

Only applies to SPMD. Number of latitudes must be an even number.
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.

FLDLST: cant find auxiliary field aaaaaa

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

FREEMEM: Bad return code = eee from hpdealloc: Abort [Message]

System routine hpdealloc 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.F 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 = \text{fff}. \text{f}

The gestbl.F procedure requires that the parameter plenest (in eslookup.h) be \text{ge}. tmax - tmin + 2 (where tmin and tmax are set in esinti.F). 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.F or getavu.F then they must make sure all units are freed (call to relunit.F) when those same Fortran files are closed.

GETMEM: Bad return code = eee from hpalloc: Abort [Message]

System routine hpalloc encountered an error while attempting to dynamically allocate memory.

GETUNIT: Unit nn 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
Sec. II.F. Using CCM3

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.F must be set to be .le. 40 to avoid this error.

HERXIN: Fatal error: ppdy must be set to 4

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

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

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

INITAL: Error returned from RDHDR for record n

An error was returned when reading header record 'n' from the initial dataset.

INITAL: Failed to open unit uu ierr=nn

The Fortran open statement returned a status of nn when opening file NSVSN (the restart dataset).

INITAL: Illegal format type on initial dataset.

The rightmost two digits of the initial dataset format type (MFTYP) must be equal to 43 (CCM3).

INITAL: MISMATCH BETWEEN INITIAL DATASET HEADER AND PARAMETERS

MLON: nnnn PLON: mmmmm
MOREC: nnnn PLAT: mmmmm
MLEV: nnnn PLEV: mmmmm
MTRM: nnnn PTRM: mmmmm
MTRN: nnnn PTRN: mmmmm
MTRK: nnnn PTRK: mmmmm

The header from the initial dataset has different resolution parameter(s) than the Model parameters indicate.
INITIAL: the regeneration file size is too small:
Maximum file size (mxszrg): \( nn \)
Size of a latitude record: \( nn \)

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

INITIAL: too many primary [secondary] regeneration files required:
Maximum number of files
\((\text{pnrg1[2]}) = \text{nnn} \)
Maximum size of files (mxszrg) = \( mmm \)
Actual number of files required = \( nn \)
Actual size of latitude record = \( 111 \)
Number of latitudes per file = \( nn \)

Either the \textit{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 \textit{namelist}.

INITEXT: sicthk field does not match initial ice distribution.

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.

INITCOM: \textit{mmax=ptrm+1 .gt. plon/2}

The spectral resolution \textit{parameters} are inconsistent.

INTHT: Too many history tapes declared, \textit{max= n}
To increase, change parameter ptapes.

ptapes defaults to 6, including the primary history file.

INTHT: Unable to open file sunit: \textit{ios= eee}

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

IOSTOP: I/O ERROR \textit{Message}

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

LAGYIN (or LAGZIN): Error: \textit{ppdy (or ppdz).ne. 4}

\textit{parameter} ppdy (ppdz) in parslt.h must be set to 4 for the semi-Lagrange code to run properly.
LIMDX: Local work array DELI not dimensioned large enough. Increase local parameter pbpts to iiiii

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

LSMINI: Need to set init data file name

When the LSM namelist variable RDLSF is set to .TRUE., the user must also specify a LSM initial dataset name via the FINIDAT variable in the LSM namelist.

LSMINI: Need to set srf data file name

User must specify a LSM surface data type boundary dataset via the FSURDAT variable of the LSM namelist.

LSMMAP fatal error: discrepancy in subgrid points kpt in lsmpar.h = mm but k in ls mmap = nn

There is a discrepancy between the resolution parameter KPT defined for the Land Surface Model in preproc.h and the values contained in the LSM surface data type boundary dataset given in the LSM namelist.

LSMMAP fatal error: discrepancy in land points lpt in lsmpar.h = mm but l in ls mmap = oo

There is a discrepancy between the resolution parameter KPT defined for the Land Surface Model in preproc.h and the values contained in the LSM surface data type boundary dataset given in the LSM namelist.

LSMMAP fatal error: weights do not sum to 1 lon = ll lat = mm : sum = nn

There is a discrepancy between the resolution parameter KPT defined for the Land Surface Model in preproc.h and the values contained in the LSM surface data type boundary dataset given in the LSM namelist.

LSMTCI fatal error: discrepancy in subgrid points kpt in lsmpar.h = kk but k in lsmtci = ll

There is a discrepancy between the resolution parameters defined for the Land Surface Model in preproc.h and the values contained in the LSM surface data type boundary dataset given in the LSM namelist.

LUNITS: Unable to open file uu: ios= eee

Error status eee was returned when opening unit uu for direct access.
MKSLIC: Level mismatch for field: ffffffff The array is single level but the data is multilevel
MKSLIC: Level mismatch for field: ffffffff The array is multilevel but the data is single level
MKSLIC.F was called incorrectly for this field, or the data being transferred are incorrectly formatted.

MKSLIC: The required input field ffffffff was not on the initial data file The available fields are: (list of fields)
The initial or boundary dataset is missing a required field.

NAVU: Ran out of Fortran unit numbers
Routine navu.F cannot find an unused logical unit number between 1 and 99.

OUTFLD: Bad tens digit in mflsd= t.
Subroutine outfld.F encountered an invalid value in the tens digit of mflsd(1). Valid values are 0 (instantaneous), 1 (averaged), 2 (minimum), and 3 (maximum). Check settings in subroutine bldfld.F.

OZNINI: All data on ozone dataset is beyond current date
Data on ozone dataset do not bracket the current Model date. The most recently read ozone header will be printed for additional information.

OZNINI: Consecutive rewinds requested on ozone dataset.
Rewinds should be requested on this dataset only to cycle for another year. This error may indicate a corrupted or empty ozone dataset.

OZNINI (or OZNINT): Error>>>>>>Two consecutive time slices on ozone dataset are more than 45 days apart.
Although the Model can run with irregularly spaced ozone data, these data are interpolated in time, so large gaps may cause inaccuracy. The most recently read ozone header will be printed for additional information.
OZNINI (or OZNINT): Multiyear file assumed. EOF read.

An unexpected end of file was encountered on a noncycling ozone dataset.

OZNINI: Ozone dataset is NOT a type 43 dataset.

The ozone dataset has the wrong format type. The CCM3 ozone dataset is a hybrid coordinate dataset.

POINTS: Mismatch between calculated length and parameter NLNBUF1 (or NLNBUF2) = nnnnnn, PFLENB (or PFLENA) = mmmmmm

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

PREALC: Cannot allocate nnnnnn bytes for file fffff, status word = m

This is not a fatal error. The attempt to preallocate disk space for the indicated file was unsuccessful. The file will still be written.

QNEG3 from NNNNNN: m=mmm lat=111 Min. mixing ratio violated at iiii points. Reset to r.rrrrrEee
Worst = w.wwwwwwEee at i,k=iiii kkk

This nonfatal message advises the user 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.

RADOZN: Bad ozone data: non-monotonicity suspected

Ozone boundary dataset may be corrupted.

RDHARR: End of File on unit uu

An unexpected end of file was encountered on unit uu

RDHARR: end of file on unit nn

An end of file was encountered in reading a history file data record. This error may occur when reading the initial dataset or any boundary dataset. See the accompanying traceback for the calling routine.
RDHDR: integer (or character or real) header record too long. Allocated space = nnn, Actual length = mmm

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.

READARR: I/O error
READARR: End of file

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

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

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

READRG: Must change case name on a branch run.
Previous name = cccc, current case name = cccc

To avoid overwriting existing output files, the case name ($CASE) 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.F is called to further explain the error message.

RELUNIT (or RELAVU): Attempt to return out of range unit=uu

Unit numbers must be in the range 1 to 99.

RESTRT: Error returned from RDHDR for record n

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: No restart data available

A restart dataset was not specified in the namelist of the previous run. Therefore, no regeneration datasets were written.

RESUME: nstep (starting time step) .ge. nstop nstep = nnn, nstop = mmm

namelist variable NESTEP should be greater than the restart timestep.
RGNFLS (or RGNHBF): Failed to open file *fff*, ierr= **eee**

Subroutine rgnfls.F or rgnhbf.F was unable to open one of the regeneration datasets. The I/O status value is printed for more information.

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

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 primary [secondary] regeneration files required:
- Maximum number of files (pnrg1[2]) = **nnn**
- Maximum size of files (mxszrg) = **mmm**
- Actual number of files required = **nn**
- Actual size of latitude record = **lll**
- Number of latitudes per file = **nn**

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.

SAVDIS: Error from mswrite -- **errmsg**

System routine mserro is called to interpret a nonzero (error) status from mswrite.

SAVDIS: Error in ishell call, ier= **nnn**

The indicated status was returned from system routine ishell. The call which generated this error is printed just prior to this message.

SCAN1BC: Failed to open file *ffff* ierr= **ee**

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.

SOMINI: Consecutive rewinds requested on sst

Rewinds should be requested on this dataset only to cycle for another year. This error may indicate a corrupted or empty sst dataset.
SOMINI (or SOMINT): Error>>>>>Two consecutive time slices on sst dataset are more than 45 days apart.

Although the Model can run with irregularly spaced sst data, these data are interpolated in time, so large gaps may cause inaccuracy. The most recently read sst header will be printed for additional information.

SOMINI (or SOMINT): Multiyear file assumed. EOF read.

An unexpected end-of-file was encountered on a noncycling SST dataset.

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 11

Internal error in PVM routines in SOMQAD. Only applies to message passing version of CCM3

SPHDEP: ***** MODEL IS BLOWING UP *********

Routine SPHDEP has recognized unreasonable values for Model winds.

SRFTSB: Error returned from mtdlss

An error has been detected in the multiple tridiagonal linear system solver mtdlss.F. 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.F for additional detail.

SRFTSBI: error returned from mtdlss

An error has been detected in the multiple tridiagonal linear system solver mtdlss.F. 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.F for additional detail.

SSTINI: All data on sst dataset is beyond current date

Data on SST dataset does not bracket the current Model date. The most recently read SST header will be printed for additional information.
SSTINI: Consecutive rewinds requested on sst dataset.
Rewinds should be requested on this dataset only to cycle for another year. This error may indicate a corrupted or empty SST dataset.

SSTINI (or SSTINT): Error >>>>>> Two consecutive time slices on sst dataset are more than 45 days apart.
Although the Model can run with irregularly spaced SST data, these data are interpolated in time, so large gaps may cause inaccuracy. The most recently read SST header will be printed for additional information.

SSTINI (or SSTINT): Multiyear file assumed. EOF read.
An unexpected end of file was encountered on a noncycling SST dataset.

STEPON: *** Running out of CPU time -- terminate ***
Subroutine stepon.F 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.

TIREAD: End-of-file reading header for time-invariant dataset, record number n
Probably the time-invariant dataset is empty or corrupted.

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.

VRTMAP: **** Error: Not enough artificial grid intervals. Currently, "pmap" is set to pppppp
Reset parameter "pmap" to at least mnnnnn
In the semi-Lagrangian transport scheme, the artificial vertical grid declared to map 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.F local parameter pmap.

WRAPUP: Failed to open file hhhhhh ierr=nnn
Auxiliary files are opened to append by wrapup.F. This open failed.
WRITERIC: bad write of record rr to unit uu

The Fortran direct access write to unit uu returned an error. iostop.F is called to further explain the error message.

WRTARR: I/O error

iostop.F is called to explain an error returned from an array write.

WRTRSL: Failed to open file fffff

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.
G. Using the Slab Ocean Model (SOM)

CCM3 is normally run with prescribed sea surface temperatures. It may also be run with the Slab Ocean Model (SOM), a thermodynamic model of the ocean mixed layer. A user familiar with running CCM3 with prescribed SST’s can easily use SOM.

There are three basic changes needed to run CCM3 with SOM: (1) enabling SOM with the setup script before building the Model executable; (2) using a special SOM initial dataset; and (3) using a special SOM time-varying boundary dataset, in place of the prescribed SST dataset.

To enable SOM when building the executable, the token COUP_SOM in the params.h file must be defined (if using the setup script to build the executable, this is done automatically if the user requests SOM). Also, the appropriate SOM files must be included in the executable (again, this is done automatically if the user requests SOM when using the setup script).

To run the CCM3 executable with SOM, a special initial dataset must be used. This dataset contains all the standard initial fields, with the addition of SNOWH (snow liquid water equivalent depth, in meters), and SICTHK (sea ice thickness, in meters). Also, a special time varying boundary dataset is required, containing twelve monthly-mean fields of ocean mixed-layer depth (MLD, in meters), and ocean heat flux \( Q_0 \) (in \( W/m^2 \)). Refer to “Slab Ocean Model (SOM) Datasets” on page 49 for a detailed description of the SOM datasets.

The output of CCM3 with SOM enabled is the same as a run with prescribed sea surface temperatures, except that the field SICTHK is added to the output history files. In addition, the field TS1 over ocean now represents the computed ocean mixed-layer temperature, or the top level sea ice temperature (when sea ice is present), with TS2, TS3, TS4 the lower layer sea ice temperatures, SNOWH the snow cover on sea ice, and SICTHK the sea ice thickness.

Apart from the changes just mentioned, the running of CCM3 with SOM is exactly like that with prescribed SST’s. Thus, the discussion on building and running CCM3 in Sec. II.A through II.F is applicable to using SOM also.
H. Getting Help

The NCAR Climate and Global Dynamics Division has historically provided complete documentation of its community tools. The CCM Core Group staff can only respond to queries involving documentable errors (bugs) in the software. 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@ncar.ucar.edu. Otherwise, the user should rely upon the available documentation or ask the question on the ccm-users e-mail group. When reporting a suspected bug, please include the following information: 1) the architecture on which the code was built; 2) horizontal resolution; 3) configuration (e.g., Slab Ocean Model); 4) namelist; 5) run script; and 6) Model output.

(a) 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.

(b) 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) and the CCM Processor. 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.

The ccm-users mail group is moderated solely to filter out annoying commercial advertisements. All messages related to the CCM will be sent to all other members of the ccm-users group. Since the members of this mail group are professionals whose time is very valuable, please make a concerted effort to keep the subjects on topic, the volume low and the quality high. The quality of this mail-group can only be maintained by the efforts and personal integrity of the individual members.

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

To SUBSCRIBE, send mail to MajorDomo@ncar.ucar.edu with the words subscribe ccm-users as the complete mail text. You will receive confirmation via e-mail and you will be automatically subscribed to the list.
To UNSUBSCRIBE, send mail to MajorDomo@ncar.ucar.edu with unsubscribe ccm-users as the complete mail text.

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

To CHANGE YOUR EMAIL ADDRESS: Simply unsubscribe from your old address and resubscribe from your new address. This prevents error messages and the necessity to manually verify your address.
III. CCM3 Internals

In this section, the CCM3 code is described in detail for those users who seek to modify the Model or add their own physical parameterizations. A brief discussion of the design philosophy is first presented, followed by an overview of the Model code flow, then a description of the data structures, including grid-point and spectral arrays. The implementation of multitasking is then discussed. Next, details of the CCM3 input and output datasets are described, with particular emphasis on their file structure. The final section of this chapter concerns disk and Mass Store file management. Familiarity with advanced aspects of the UNICOS operating system is assumed. A glossary of terms is presented in "Appendix A: Glossary."

A. Design Philosophy of CCM3

Large computer models are notorious for being difficult to read, understand, and modify. In 1989, an international committee of geophysical modelers wrote some coding rules designed to improve these aspects of large models, especially when adding new parameterizations. 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. Since the CCM3 Model is essentially a modified version of CCM2, continued use of these coding rules was employed.

The CCM3 code has retained the simplicity of use that was established in CCM2 and previous versions of the Community Climate Model. Thus, the input and output datasets and namelist input of CCM3 are very similar to those of CCM2. The control structure of CCM3 with respect to initialization, time-stepping and grid-point to spectral transformations remains similar to that of CCM2. However, some significant changes were made to the CCM2 code in going to CCM3, in order to allow a straightforward coupling of the atmospheric component of the Model with the land, sea, and ice components.

Modifications were also made in order to allow the code to be run on other computer architectures than the Cray Y-MP. Currently, the Model can be run on the Cray Y-MP, Cray T3D, IBM RS6K, and SUN architectures. Other significant changes to the CCM2 code are summarized in "Historical Development of the CCM3" on page 1.

The memory management technique implemented in CCM2 represented a significant change from the previous version of the Community Climate Model, CCM1. The double-buffering of data in CCM1 was replaced with the use of the fast-access solid-state storage device (SSD) on the Cray Y-MP. Thus, in CCM2, the main Model buffers were stored out-of-core on the SSD (out-of-core refers to Model data structures which reside in secondary storage and are cycled in and out of main memory as required).
CCM2’s use of the SSD was optimized by synchronous unblocked I/O to Secondary Data Segments (SDS). CCM3 has retained the out-of-core features of CCM2 memory management, allowing it to be run at high-resolution on small memory machines, but has also added the capability of retaining part or all of the buffers entirely in-core (as dictated by namelist variables INCORHST, INCORBUF, and INCORRAD). As in CCM2, the main Model buffer of CCM3 consists of only those variables that must be carried at more than one time level, more than one latitude scan, or for 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 in doing I/O for the out-of-memory implementation.

To accommodate the semi-Lagrangian transport code, certain variables need to be retained in main memory as three-dimensional arrays (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. Named common blocks are also used to hold variables that do not vary latitudinally, as a means of avoiding enormous argument lists. All data statements which set common variables are contained in block data subprograms.

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 detail in “Data Structures” on page 103.

Within the time-stepping loop of stepon.f, only the control routines linemsbc.f, linemsmac.f, and spegrd.f interface directly into the buffers. At the beginning of these routines, Cray pointers are used to associate fields in the main Model buffers with mnemonically named field arrays. The physical parameterization routines called by these control routines need only pass these mnemonic fields through argument lists, without the hassle of dealing with the clumsy indexing of the buffers. Thus, physical parameterization packages can easily be replaced by other packages, i.e., “plug-compatible.”

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 reproducible results.

The CCM3 history file interface is designed to provide users with easily used utility routines for recording data on the output history files. Fields that appear in a Master Field List, generated at initialization time by subroutine bldfld.f, may be included on the history file either by default or by namelist request of the user. A field is recorded in the history file buffer during the time-stepping phase by calling subroutine outfld.f.

The user is responsible for keeping track of the time at which the recorded field is valid and how it relates to the time recorded on the header. Provision is made for recording both instantaneous and averaged, as well as maxima and minima. Fields are written to the disk history file within a multitasked loop over latitudes. Due to the multitasking and the resulting random order of execution of the first latitude loop, the history file (which is written to sequentially) may be randomly ordered by latitude. Thus, each latitude record contains as its first word a latitude index, with indices increasing from south to north.

The CCM3 code follows a written syntactical standard, described in “CCM3 Coding Standard” on page 164 of this document. The code has been edited to appear homogeneous as far as comments, indenting, etc., to make it more readable to the user. The intent of the “commenting” standard is to internally document the code so as to make a separate module document unnecessary.
B. Model Code Flow

This section provides a narrative of the CCM3 code flow through the initialization phase and the driving computational loops (time-stepping phase). The graphical calling tree illustrated in "Appendix B: CCM3 Calling Tree" on page 175, provide helpful reference for reading this section. The user may also wish to review “Overview of the Structure of CCM3” on page 4 before reading this section. Details of the physical parameterizations in CCM3 are not presented here — see Kiehl et al. (1996).

1. Initialization

The first tasks performed in a CCM3 run are defining the logical units used for I/O (subroutine lunits.F), presetting namelist variables, (subroutine preset.F) reading in the Fortran namelist data (subroutine data.F), and setting pointers and lengths associated with the main Model buffer (subroutine points.F).

From this point on, initialization takes one of two paths: initial runs are set-up in subroutine initial.F, or continuation runs are initialized in subroutine resume.F. These two paths are roughly parallel in nature and call many of the same routines. The primary difference is that in an initial run, the initial dataset must be read, and prognostic fields spectrally transformed (see subroutine inidat.F, below). In a continuation run these fields are obtained from the appropriate regeneration files.

The primary job of inidat.F is to read in the required fields from the initial dataset, and to 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 subsurface temperatures (TS1 through TS4), and the moisture field (Q3). If the slab ocean option 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.F is quite large. Three-dimensional arrays exist for vorticity, divergence, and the work array required by the fft package. This 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” on page 117), the large size of the local database does not increase total memory utilization by CCM3 since the local workspace in inidat.F will disappear before the time integration begins.

2. Time Integration

Subroutine stepon.F controls the time-stepping in CCM3. After the initialization phase is complete for either an initial run or a continuation run, stepon.F loops through the time integration to completion of the simulation, incrementing /comtim/ variable nstep after each timestep. If the
run terminates normally, `scan1bc.F` will be the last routine called, as well as the first to be executed upon restart. The only reason to call `scan1bc.F` at the end of a run is to write the fully time-filtered data to the history file, completing a time accumulation period. Thus, the history file buffer need not be written to the regeneration dataset.

Subroutine `advance.F` is called from `scan1bc.F` for each timestep, outside of the multitasked latitude loop. `advance.F` drives the updating of current time information (`caldyi.F`) and time-variant boundary dataset information (`sstint.F` or `somint.F` and `oznint.F`).

Current simulation time information is computed in `caldyi.F`. From the standpoint of Model computations, the output variable `calday` (current Julian day plus fraction) is all that is required. Other variables having to deal with the current date and current day are also output from `caldyi.F`, because they are required by the history file handler. Note that the definition of a “year” is 365 days. There are no leap years.

(a) Latitude Scans

There are two multitasked, Gaussian latitude scan loops: one in routine `scan1bc.F` and another in `scan1ac.F`. These loops used to be just one routine, `scan1.F`, in earlier versions of the Community Climate Model. This routine was split into two pieces in CCM3 in order to accommodate coupling to other geophysical models, such as ocean models, sea ice models, and land models. The “bc” in “scan1bc.F” refers to “before coupling”, and the “ac” in “scan1ac.F” 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 by the Model between calls to `scan1bc.F` and `scan1ac.F`.

Subroutine `lsmi.F` 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. The computations associated with each of these subsections are then distributed among the available processors. Refer to description of LSM output on page 69.

Subroutine `somoce.F` drives the slab ocean model calculations, if they are enabled. Enabling this option requires `cpp` token `COUP_SOM` to be set with a `#define` in `params.h` (done automatically by the setup script described in Sec. II.A). For more details about the Slab Ocean Model see “Using the Slab Ocean Model (SOM)” on page 91.

Subroutine `scanslt.F` drives the semi-Lagrangian transport calculations. This is the only latitude scan in CCM3 in which the full dimensionality of the “extended grid” (`plond` and `platd`) is required. In addition to the physical data locations (`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 sltint.f.

In scan1lt.f, the un-time-filtered wind field at time level “n” is used to predict the evolution of the time-filtered moisture and (if applicable) constituent fields, from time n-1 to time n+1 using the semi-Lagrangian transport scheme (Williamson and Rasch et al., 1989). Forecast expense in terms of computer time for additional constituents is approximately a 5% increase in CPU seconds per constituent (standard T42 resolution, spectral dynamics). Thus, adding 10 constituents would result in a 50% increase in CPU time for the run. No I/O is done in scan1lt.f, since the prognostic arrays u3, v3, wfid, and q3 are entirely in-core.

The local data structure in scan1lt.f 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. The global data increases significantly as well.

Routine sltini.f 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.f, but the routine is multitasked over latitude bands since it is called every timestep.

The primary function of scan2.f is as a driving routine for the conversion of spectral space prognostic variables back to grid-point space, and computation of global integrals of mass for the timestep. The multitasked latitude loop within scan2.f is over latitude pairs rather than individual latitudes in order to take advantage of the symmetric properties of spectral coefficients for north-south Gaussian latitude pairs. The last step in scan2.f is to toggle the time indices n3 and n3ml after the conversion back to grid-point space is complete. No copying of the actual data is necessary. The prognostic data that were pointed to by the current time index (n3) will become the previous time index (n3ml), and the data just computed in time index n3ml will become the “next” (n3) time index at the start of the next iteration in stepon.f.

(b) Spectral Space Computations

Gaussian quadrature (quad.f), completion of the semi-implicit timestep (tstep.f), and horizontal diffusion calculations (hordif.f) are all accomplished within multitasked loops in dyndrv.f. The quadrature and semi-implicit timestep computations are parallelized over total wavenumber “n” when the architecture type is CRAY, and over Fourier wavenumber “m” on other architectures for which the “SPMD” option has been enabled. Horizontal diffusion calculations are parallelized over the vertical index “k”.

Model Code Flow
(c) Computational Driving Routines for Each Latitude Band

Routines `linemsbc.F` and `linemsac.F` sit beneath the multitasked latitude loops in `scanlbc.F` and `scanlac.F`, respectively. Various functions of these “kitchen sink” routines include allocation of memory for the main Model buffer and history buffer, application of the SLT “fixer”, time-filtering, and calling the adjustment and tendency physics driving routines.

Physical parameterizations which are applied as hard adjustments to the prognostic variables rather than computed as tendencies are driven by subroutine `aphys.F`. These parameterizations include dry adiabatic adjustment, moist convection, and large-scale condensation.

Routines `tphysbc.F` and `tphysac.F` (called from `linemsbc.F` and `linemsac.F`, 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, rayleigh friction, and gravity wave drag. Cloud calculations are needed internally only by the radiation code. Since radiative computations are normally not performed on every timestep, clouds calculations are performed only on timesteps that the radiative transfer code is exercised.

Subroutine `radctl.F` controls both the longwave and shortwave radiative transfer code. While the rest of the Model uses mks units, this routine uses cgs units. The timesteps on which the longwave absorptivity and emissivity calculations are performed is controlled by namelist variable `IRADAE`. This variable must be an even multiple of `IRADLW` (that is, the absorptivities and emissivities can be calculated only on a radiation timestep). In addition, `IRADAE` should be an even multiple of the history file write frequency, `NHTFRQ` (1), to ensure that the absorptivities and emissivities need not be written to the secondary regeneration dataset.

If `IRADAE` is greater than `IRADLW`, then absorptivity and emissivity (a/e) values are stored in a buffer between successive radiation calculations. These absorptivities and emissivities can be stored out-of-core or in-core. If on a timestep in which the a/e calculations are performed, these data are written to work unit `nabem` if out-of-core, or to buffer `bigbufc` if in-core (see `coreiorad.h`). If on a non-a/e radiation timestep, and if the a/e data are stored out-of-core, then the a/e are read in from out-of-core unit `nabem`. Control of this computation and I/O is done in `radclw.F`, which is called from `radctl.F`.

Routine `sltb1.F` 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).
C. Data Structures

CCM3 in its default configuration is a fully in-core Model. 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 buffer, the radiation buffer, and the history buffer. For the in-core implementation, I/O calls to and from the SSD are replaced by the setting of Cray pointers to the address of the relevant latitude band of each in-core data structure.

This section is devoted to a discussion of the design philosophy and implementation techniques for both in-core and out-of-core grid-point and spectral space data structures in CCM3. Some special array-indexing constructs are also addressed.

1. Vertical Coordinate

The vertical coordinate used in CCM3 is a hybrid sigma-pressure system. In this system, the upper regions of the atmosphere are discretized by pressure only. Lower vertical levels have the terrain-following sigma \((p/ps)\) vertical coordinate smoothly merged in, with the lowest level(s) being pure sigma. A schematic representation of the hybrid vertical coordinate and vertical indexing used in CCM3 is presented in Figure III.1.

2. Grid-point Data Structures

Grid-point space prognostic variables and many arrays associated with the semi-Lagrangian transport scheme (SLT) are maintained in-core. Other grid-point fields exist either in one of the three Model buffers (main Model buffer, radiation buffer, history buffer), or as local workspace in individual routines. All grid-point data are stored with longitude as the fastest varying (inner most) 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 species. The number of constituents, defined by the Fortran parameter “\texttt{pcnst},” will always be at least one since water vapor is always transported in the default Model configuration. The variable \texttt{lat} is used consistently throughout the CCM3 code to represent the current latitude index. Two time-level indices are necessary for the prognostic variables due to the leapfrog time-stepping scheme.
Grid-point space arrays are almost universally declared with longitude dimension \texttt{plond}, where \texttt{plond} = \texttt{plon} + 1 + 2*\texttt{nxpt}. The parameters \texttt{plond}, \texttt{plon}, and \texttt{nxpt} are all defined in routine \texttt{pmgrid.h}.

Parameter \texttt{plon} is the number of actual data points in the longitudinal direction, and \texttt{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 by the \texttt{fft} package. 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, it was decided to use \texttt{plond} wherever possible, whether or not it was 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 grid-point space routines. It is undesirable to have to keep track of where the physical data start for 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.F from scanlbc.F (this is very high in the calling tree: refer to Figure B.2 on page 178). Some of the calling and declaration sequence looks like:

```
scan1bc.F:
```

```fortran
    call linemsbc(..., ps(1, lat, n3ml),
               $u3(i1, l, j, n3), ...)
```

```
linemsbc.F:
```

```fortran
    subroutine linemsbc(...psml, u3,...)
    real psml(plond),
    $    u3(plond, plev)
```

In scan1bc.F, each argument passed to linemsbc.F 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 $u3$). This way all references to fields in linemsbc.F and below in the calling tree can begin at array element 1. The user need not be concerned with offsets unless they are dealing with code at the scan1bc.F level or higher up in the calling tree where the full data structures are available.

Analogous to the longitudinal case, additional nonphysical grid points 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 Model 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.F passes to lower-level routines the array address corresponding to the latitude index at which the physical data start.

In the above example, at the level of linemsbc.F, latitude indices are no longer present because all computations in linemsbc.F and below in the calling tree 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.

(a) Model Buffer

The main Model buffer contains longitude-level slices of a number of 
fields stored contiguously. In an out-of-core configuration, these arrays are 
cycled to and from an I/O device, latitude band by latitude band. A sche-
matic representation of this buffer is presented in Figure III.2.

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 radia-
tion routines.

Surface pressure, vorticity, and divergence are in the buffer for contiguity 
purposes. Contiguity is required for optimal vectorization of the fft pack-
age. This is important since the fft routines used in CCM3 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. $2 \times 18 + 1$) 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 actually memory-contained (common/com3d/), and is copied 
to and from the Model buffer for the sole purpose of avoiding the fft per-
formance 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 (cft77 -Rb on Cray machines). This 
functionality is obtained through Cray Fortran pointer variables, as illus-
trated in a section of subroutine spegrd. F below:

```fortran
real b2(plenalc1)  ! 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 the Model buffer, can be referenced as a sepa-
rate two-dimensional array once the Cray pointer is set. Though not part of
### Data Structures

#### Buffer

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_1(pflenb) )</td>
<td>( nztl )</td>
<td>Vorticity: half time-filtered ((m))</td>
</tr>
<tr>
<td></td>
<td>( ndtl )</td>
<td>Divergence: half time-filtered ((m))</td>
</tr>
<tr>
<td></td>
<td>( nphis )</td>
<td>Surface geopotential ((s))</td>
</tr>
<tr>
<td></td>
<td>( nqminus )</td>
<td>( q ) being advected in slt code ((m))</td>
</tr>
<tr>
<td></td>
<td>( npsm2 )</td>
<td>Surface pressure, time ( n-2 ) ((s))</td>
</tr>
<tr>
<td></td>
<td>( num2 )</td>
<td>( u )-wind, time ( n-2 ) ((m))</td>
</tr>
<tr>
<td></td>
<td>( nvm2 )</td>
<td>( v )-wind, time ( n-2 ) ((m))</td>
</tr>
<tr>
<td></td>
<td>( ntm2 )</td>
<td>Temperature, time ( n-2 ) ((m))</td>
</tr>
<tr>
<td></td>
<td>( nqm2 )</td>
<td>Constituents, ( H_2 O ) first ((m))</td>
</tr>
<tr>
<td></td>
<td>( npblht )</td>
<td>PBL height ((s))</td>
</tr>
<tr>
<td></td>
<td>( nqpert )</td>
<td>PBL moisture/constit. perturb. ((s))</td>
</tr>
<tr>
<td></td>
<td>( ntpert )</td>
<td>PBL temperature perturb. ((s))</td>
</tr>
<tr>
<td></td>
<td>( nfsns )</td>
<td>Surface absorbed solar flux ((s))</td>
</tr>
<tr>
<td></td>
<td>( nqrs )</td>
<td>Shortwave rad. heating rate ((m))</td>
</tr>
<tr>
<td></td>
<td>( nqr1 )</td>
<td>Longwave rad. heating rate ((m))</td>
</tr>
<tr>
<td></td>
<td>( npsm1 )</td>
<td>Surface pressure, time ( n-1 ) ((s))</td>
</tr>
<tr>
<td></td>
<td>( nzm1 )</td>
<td>Vorticity, time ( n-1 ) ((m))</td>
</tr>
<tr>
<td></td>
<td>( ndm1 )</td>
<td>Divergence, time ( n-1 ) ((m))</td>
</tr>
<tr>
<td></td>
<td>not used</td>
<td>(m)</td>
</tr>
<tr>
<td></td>
<td>not used</td>
<td>(s)</td>
</tr>
<tr>
<td></td>
<td>not used</td>
<td>(s)</td>
</tr>
</tbody>
</table>

#### Buffer

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_2(pflena) )</td>
<td>( npspl )</td>
<td>Sfc. pressure, time ( n(s))</td>
</tr>
<tr>
<td></td>
<td>( nzp1 )</td>
<td>Vorticity, time ( n(m))</td>
</tr>
<tr>
<td></td>
<td>( ndp1 )</td>
<td>Divergence, time ( n(m))</td>
</tr>
<tr>
<td></td>
<td>( ndthp1 )</td>
<td>Horiz. temp. diffusion ((m))</td>
</tr>
<tr>
<td></td>
<td>( ndpsl1p1 )</td>
<td>Long. deriv. of lpns, time ( n(s))</td>
</tr>
<tr>
<td></td>
<td>( ndpamp1 )</td>
<td>Lat. deriv. of lpns, time ( n(m))</td>
</tr>
<tr>
<td></td>
<td>( nduhp1 )</td>
<td>Horiz. ( u )-momentum diff. ((m))</td>
</tr>
<tr>
<td></td>
<td>( ndvhp1 )</td>
<td>Horiz. ( v )-momentum diff. ((m))</td>
</tr>
<tr>
<td></td>
<td>( ndpspl )</td>
<td>Pressure gradient ((s))</td>
</tr>
</tbody>
</table>

**Figure III.2.** Layout of main Model buffers, \( b_1 \) and \( b_2 \). 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 \( pflenb, pflen, pbflnb, pbflna, pfdalcl \).

The ANSI Fortran standard, Cray pointers are supported on a wide variety of platforms.

#### (b) History Buffer

Data written to the history files are maintained in a separate buffer distinct from the main Model buffer. This buffer may be cycled to the SSD (unit \( sunit \)) or maintained in-core. In either case its existence must span time levels because most fields written to the history file are time-averaged. Therefore, these data must be carried across time levels between the
outfld.F 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 “History Buffer” on page 132 for more details.

(c) Absorptivity/Emissivity Arrays

Like the main Model buffer and the history buffer, 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 (specified by namelist variable NHTFRQ(1)) that divides evenly into the frequency of the longwave radiation calculation (IRADAE). Otherwise, the large absorptivity/emissivity arrays must be written to the regeneration dataset.

(d) Out-of-Core Data Storage: The SSD Work Units

Data flow between Model arrays and the out-of-core units is depicted in Figure III.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 multilayered 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. Note, nra1, nrbl, and nabem are the Fortran unit numbers associated with the out-of-core work units. On start-up, subroutine inidat.F does the initial copy of data from the main Model buffer to units nra1 and nrbl. Data flow between the history buffer and out-of-core work unit unit is analogous to the main Model buffers. All I/O in CCM3 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 machine architectures an easier task. On Cray machines, CCM3 uses the UNICOS “Secondary Data Segment” (SDS) file construct 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 in CCM3 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.
Data Structures

**Figure III.3.** Use of SSD Work Units in the CCM3 out-of-core configuration. Arrows indicate data movement between Model buffers (vertical rectangles) and the out-of-core work 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 labelled with the subroutine name responsible for the data transfer.

**(e) Local Workspace**

Nearly all local workspace declared in individual routines is stack-based. That is to say, 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 "CCM3 Multitasking Strategy" on page 115.

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.
(f) **In-Core Grid-point Arrays**

Prognostic variables surface pressure, zonal wind, meridional wind, temperature, and water vapor (hereafter referred to by their CCM3 variable names $p_s$, $u_3$, $v_3$, $t_3$, and $q_3$ 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 core memory.

A schematic view of the data structure defining $u_3$, $v_3$, $t_3$, and $q_3$ is shown in Figure III.4. This perplexing storage arrangement accomplishes two important goals. The contiguity of fields $u_3$, $v_3$, and $t_3$ 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 $u_3$ and $v_3$ to produce more efficient code.

![Figure III.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 code.](image-url)
If $	ext{pcnst}$ is greater than 1 (that is, at least one user-defined constituent is being transported), then 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 

$q_3(i1+i-1, k, m, j1+ \text{lat}-1, n3)$, where $m$ is the constituent index, \text{lat} is the south-to-north latitude index, and $n3$ is the time level index.

Variable $m=1$ refers to the moisture field, with additional constituents stored in locations $m=2$, PCNST. Variables $i1$ and $j1$ are the longitudinal and latitudinal offsets into the array required by the SLT code. As described earlier, much of the ugliness involved in accessing elements of this data structure can be hidden by passing in 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. Fortran parameter \text{plevd}, defined as \text{plev}*(3+pcnst), allows dimensioning of $u3$, $v3$, and $t3$ as $(\text{plond}, \text{plevd}, \text{platd}, 2)$.

The \textit{common} block in which the prognostics reside is declared as follows:

```fortran
common/com3d/n3ml,n3
common/com3d/ps(plond,plat,2),x(plond,plev,3+pcnst,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,3+pcnst,platd,2) ! specific humidity and constituents
  equivalence (u3,x(l,,1,1,1)),
$ (v3,x(1,l,2,l,l)),
$ (t3,x(l,1,3,1,1)),
$ (q3,x(1,l,4,1,1))
C
integer n3ml,n3 ! time index pointers
real ps, ! surface pressure
$ x ! contiguous u,v,t,q
```

The 3 in the 3+pcnst dimension accounts for prognostic variables $u3$, $v3$, and $t3$. \textquotedblright n3\textquotedblright and \textquotedblright n3ml\textquotedblright are time level pointers 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 \textquotedblright x\textquotedblright allows reference to all four three-dimensional prognostic arrays as a single entity.

\textit{SLT in-core arrays}

In addition to the prognostics, SLT three-dimensional fields $\text{lammp}$, $\text{phimp}$, $\text{sigmp}$, and $\text{qfcest}$ are held in-core and given global scope by keeping them in a \textit{common} block (/comslt/). These arrays are in a \textit{common} block, rather than local to scanslt.F, because they need to be writ-
ten to the regeneration dataset and because they are required in other scan loops.

Other three-dimensional arrays held in memory, but which are not written to the regeneration dataset are local scanslt F arrays ux1, uxr, qx1, and qx3. The SLT scheme may transport an arbitrary number of constituents. It is important to realize that in-core arrays q3, qfcst, qx1, and qx3 have a constituent dimension, and transporting a large number of constituents can lead to rapid growth in memory use.

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 target architecture is CRAY, 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 CCM3 spectral space arrays is presented in Figure III.6.

When the target architecture is other than CRAY, 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 III.5.

4. Other Common Blocks and Header Files: The Parameterization Interface

In keeping with the so-called “plug-compatible” coding philosophy, CCM3 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 code. 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 make up the physics of CCM3 are hooked to the Model control code using a standard form of initialization and interface routines. Each parameterization contains an initialization routine called on Model start-up. In most cases, these routines are named...
Figure III.5. Spectral storage arrangement when architecture type is not CRAY. Dashed lines depict mapping of 2-dimensional conceptual location in triangular data structure to actual linear storage in memory.

"xxinti.F," where "xx" is the standard prefix for the particular parameterization. Most of these routines are called from subroutine inti.F, before the time-stepping procedure begins. These routines set constants in the parameterization-specific common blocks and do other initialization tasks that are not time or latitude dependent. Some data initialization is done within various block data subprograms.

Parameterization common blocks

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.F. Then subroutine inti.F passes the required /comcon/ constants into each "xxinti.F" routine, which sets the individual common block constants from the passed values. It is at this point that unit conversions or other manipulations as required by the individual parameterization should be performed.

Other common blocks (i.e. those not directly related to physics code) are defined by functional purpose and exist either to give their variables global scope or to avoid the necessity of passing certain variables through many argument lists.
Each parameterization includes an interface routine that passes data between the high-level driving routines and the parameterization via the Fortran calling sequence. This approach allows the parameterization to access each required array using a mnemonically named multidimensional array, as shown in "Model Buffer" on page 106.
D. CCM3 Multitasking Strategy

CCM3 was designed to run most efficiently on the shared-memory parallel vector machines built by Cray Research, Inc. (CRI). The CCM3 developers decided early on to make the Community Climate Model a multitasked code, with the implementation details transparent to the user. To run multitasked on a Cray PVP machine, the user need only specify the number of processors to use (environment variable $NCPUS).

New to CCM3 is the capability to run multitasked on distributed memory, or so-called “massively-parallel” architectures that support the PVM message-passing programming paradigm. Currently the Cray T3D is the only supported architecture which is specifically designed for MPP applications, though the code should run on any machine or group of machines on which PVM is supported. In addition to the T3D, the message-passing code has been validated on Solaris, SunOS (SUN), and AIX (IBM) machines.

One prerequisite in the design of CCM3 was that identical simulations be guaranteed regardless of the number of processors utilized. To accomplish this goal, certain summations of critical quantities are always 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.F. The history files from two runs executed on a different numbers of processors will compare bit for bit. When adding new code, the user should always ensure that deterministic results are obtained when running multitasked. Side-by-side simulations done on different numbers of processors should be sufficient to answer this question.

1. Shared-Memory Multitasking

The CCM3 code is highly parallel. Turnaround time should therefore improve dramatically (in wall clock time) for a multitasked job, depending upon the value of $NCPUS. In a dedicated environment, $NCPUS should always be set equal to the number of physical processors on the machine.

The PVP code is multitasked using Cray “autotasking.” An advantage of autotasking over other Cray multitasking products is that it introduces no portability problems. All information required by system software to multitask the code is contained in directives which take the form of Fortran comments.

Although the autotasking preprocessor is designed as an “automatic multitasker,” CCM3 employs neither its automatic data scoping nor its automatic multitasking features. Data-scoping directives produced by the autotasker often proved to be incorrect, and the fine granularity of the automatic multitasking resulted in unacceptable system overhead. Also, identical answers could not be obtained from single-threaded vs. multitasked runs. As a result, the only features of the autotasker which are utilized in
CCM3 are those which 1) indicate which iterations of “do loops” may be run in parallel, and 2) generate system calls to parcel out work to available processors and synchronize at the end of the loop.

Autotasking directives are placed very high in the calling tree telling the Fortran mid-compiler (fmp) to autotask specific loops. Data scoping is done explicitly in these directives, which are of the form:

\[
\text{CMIC$ DO ALL SHARED (...) PRIVATE (...).}
\]

The DO ALL portion of the directive tells fmp 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 can be shared across multiple processes. Conversely, variables declared as PRIVATE must have separate storage 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.


In any multitasked code, work done by one processor must be independent of the work being done (potentially) simultaneously on all other processors. In CCM3, iterations of the Gaussian latitude loop in scanlac.F are independent of latitude. Therefore, this routine contains a multitasked loop of the form:

\[
\text{CMIC$ DO ALL SHARED(...) PRIVATE(...) do lat=1,plat ... end do}
\]

Computations within the loop for each value of lat are done independently. Work is parcelled out to available processors by the operating system 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.F and sltinini.F), each latitudinal iteration may be done in parallel. In the spectral dynamics driven by dyndrv.F, the Gaussian quadrature and semi-implicit timestep computations are parallelized over diagonals of “m-n” wavenumber space. Subroutine dyndrv.F 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, the static variables it uses must be set in a single-threaded part of the code before it is invoked.

Another result of the unpredictability of calculation order under multitasking was that direct access I/O to the work file units became necessary. If sequential access were utilized, 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.

(a) Shared-Memory Management

A stack-based memory management scheme was a crucial element in the design of the shared-memory multitasked version of CCM3. When data is allocated on the stack, its lifetime is limited to the time the subroutine or function in which it is declared is active. In other words, storage for locally defined workspace in a given routine disappears upon execution of the return statement. Stack-based local memory allocation makes for more efficient memory usage since memory is allocated to each routine only when required.

Another important reason why stack-based memory allocation is used in CCM3 has to do specifically with multitasking. 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 "re-entrant," 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 in CCM3 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 \texttt{save} statement, or by virtue of appearing in a \texttt{data} statement.

One trade off of running CCM3 on multiple processors is that memory usage is increased as compared to a single-threaded run. A small amount of overhead is created by the autotasking libraries themselves. The primary culprit, however, is the stack memory replication across processors just discussed. To find out how much memory a job actually used, run the \texttt{ja -h} command after the Model has completed execution. Output will be of the form:

```
Command  Started  ...  Memory
Name     At        HiWater
========== ========== =========
ccm3      18:11:15 ... 17520
```

The 17,520 number says that the Model used 17,520 blocks \( \times \) 512 words/block = 8,970,240 words of memory at its high-water point, i.e. the most memory that was required at any one time.

2. Distributed Memory Multitasking

The one-dimensional latitudinal data decomposition employed in the shared-memory multitasked code was also employed in the distributed-memory message-passing code. 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 III.5 on page 113). The reason is that the spectral transform technique requires summations over total wavenumber \( "n" \) which are most conveniently done on-processor. Since the loops which perform spec-
tral space computations therefore look completely different in the distributed-memory code, most of the routines which are responsible for the spectral dynamics contain "#ifdef" constructs around the entire subroutine. The only reason the shared-memory code was not rewritten to employ this same data ordering (and therefore eliminating a large number of #ifdef's) is that much longer inner loops, and therefore much higher vector performance on a PVP machine, is obtained when the data are stored along diagonals of "m-n" wavenumber space.

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.

(a) 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 parameters. The reasons for this 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 T3D 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.

(b) Distributed Memory I/O

All I/O in the distributed-memory code is done by a master processor. This includes only the history and regeneration 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 the CCM3 routines which write the history and regeneration files.
E. Details of CCM3 Input and Output Datasets

The CCM3 requires several input datasets in order to execute, and generates several output datasets during execution. The input datasets consist of the initial and boundary condition datasets. In addition the namelists CCMEXP and LSMEXP are needed to initiate a run. Output datasets include the history files, restart/regeneration files, and the Model printed output. The namelist input is covered in detail in Sec. II.B with multiple examples shown in Sec. II.C. Model printed output is discussed in detail in Sec. II.F and an example listing is shown in "Appendix C: CCM3 Printed Output." Section II.D and Section II.E describe the contents of the input and output datasets in detail, but provide no explanation of the structure of these files and how they are written. The purpose of this section is to describe the structure and contents of the initial and boundary condition datasets, the history files, and the restart/regeneration files, as well as provide an indication of where and how they are written in the code. Knowledge of the material presented in Sec. II.B through II.E is assumed.

1. History Files

The history files contain descriptive information and data values written at specified times during the run. As discussed in Sec. II.E, 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 II.9 on page 53) 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 II.9). 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 on tape. One means of viewing the contents of a history file is accomplished by using the CCM Processor (Buja, 1992). In the text that follows, a detailed description of the history file format, buffer, and code flow is presented.

(a) History File Format

Both primary and auxiliary history files are written as a series of "time samples". The structure of a history file is illustrated in Figure III.7. Notice that one history file contains MFILT time samples, each time sample consisting of three header records followed by $\text{plat}$ latitude data records ($\text{plat}$ is a Fortran parameter set equal to the number of latitude lines in 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" on page 129. The history files are written to the disk sequentially and in binary format.
Figure III.7. History file format. One disk history file contains \texttt{MFILT} time samples. Each time sample consists of three header records and \texttt{plat} latitude data records.
(b) 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 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.

The contents of the three header records are shown in Table III.1 through Table III.3. These tables present the contents of the integer header, the character header, and the real header, respectively. 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.

The output fields in a CCM3 history file are located on the hybrid coordinate grid. Thus, three arrays of hybrid coefficients are saved in the real header record along with the Gaussian latitudes and weights (see Table III.3).

**Table III.1: History File Header Record 1—Integer Variables**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENHDI</td>
<td>1</td>
<td>Length of header record 1</td>
<td>292</td>
</tr>
<tr>
<td>MFILH</td>
<td>3</td>
<td>Number of this “time sample” within this disk file or Mass Store tape.</td>
<td>1</td>
</tr>
<tr>
<td>MFILTH</td>
<td>4</td>
<td>Maximum number of time samples in this history file.</td>
<td>20</td>
</tr>
<tr>
<td>NRBD</td>
<td>5</td>
<td>Number of records before data records</td>
<td>3</td>
</tr>
<tr>
<td>MAXSIZ</td>
<td>6</td>
<td>Length of packed data record in words</td>
<td>33662</td>
</tr>
<tr>
<td>NDAVU</td>
<td>7</td>
<td>Length of the data record in words after unpacking</td>
<td>65282</td>
</tr>
<tr>
<td>8</td>
<td>Unused</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
### Table III.1: History File Header Record 1—Integer Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLON</td>
<td>9</td>
<td>Number of longitude points in one latitude line</td>
<td>128</td>
</tr>
<tr>
<td>NLONW</td>
<td>10</td>
<td>Number of longitude data values written on the history file</td>
<td>128</td>
</tr>
<tr>
<td>NOREC</td>
<td>11</td>
<td>Number of latitude lines and number of data records</td>
<td>64</td>
</tr>
<tr>
<td>NLEV</td>
<td>12</td>
<td>Number of vertical levels written on the history file</td>
<td>18</td>
</tr>
<tr>
<td>NTRM</td>
<td>13</td>
<td>M spectral truncation parameter</td>
<td>42</td>
</tr>
<tr>
<td>NTRN</td>
<td>14</td>
<td>N spectral truncation parameter</td>
<td>42</td>
</tr>
<tr>
<td>NTRK</td>
<td>15</td>
<td>K spectral truncation parameter</td>
<td>42</td>
</tr>
<tr>
<td>NFLDH</td>
<td>16</td>
<td>Number of fields in this time sample</td>
<td>85</td>
</tr>
<tr>
<td>NSTEPA</td>
<td>17</td>
<td>Timestep on which this data was written</td>
<td>0</td>
</tr>
<tr>
<td>NSTPRH</td>
<td>18</td>
<td>Iteration number for the start of this run</td>
<td>0</td>
</tr>
<tr>
<td>NITSLF</td>
<td>19</td>
<td>Number of timesteps since last tune sample was written</td>
<td>0</td>
</tr>
<tr>
<td>NDBASE</td>
<td>20</td>
<td>Base day number for this case</td>
<td>0</td>
</tr>
<tr>
<td>NSBASE</td>
<td>21</td>
<td>Base number of seconds for this case</td>
<td>0</td>
</tr>
<tr>
<td>NDCUR</td>
<td>22</td>
<td>Current day number corresponding to NSTEPA</td>
<td>0</td>
</tr>
<tr>
<td>NSCUR</td>
<td>23</td>
<td>Seconds of the current day NDCUR</td>
<td>0</td>
</tr>
<tr>
<td>NBDATE</td>
<td>24</td>
<td>Base date (yr mo day) as 6-digit integer</td>
<td>000901</td>
</tr>
<tr>
<td>NBSEC</td>
<td>25</td>
<td>Seconds to complete NBDATE date. (Note: may be different from NSBASE.)</td>
<td>0</td>
</tr>
<tr>
<td>NCDATE</td>
<td>26</td>
<td>Current date (yr mo day) as 6-digit integer corresponding to NSTEPA</td>
<td>000901</td>
</tr>
<tr>
<td>NCSEC</td>
<td>27</td>
<td>Current seconds for date NCDATE</td>
<td>0</td>
</tr>
<tr>
<td>MDT</td>
<td>28</td>
<td>Model timestep in seconds</td>
<td>1200</td>
</tr>
<tr>
<td>MHISF</td>
<td>29</td>
<td>Frequency in timesteps at which time samples are written</td>
<td>72</td>
</tr>
</tbody>
</table>
Table III.1: History File Header Record 1—Integer Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFSTRT</td>
<td>30</td>
<td>Flag to indicate type of run:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 0 for initial run</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 1 for restart</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 2 for regeneration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 3 for branch run</td>
<td></td>
</tr>
<tr>
<td>LENHDC</td>
<td>31</td>
<td>Length of header record 2</td>
<td>259</td>
</tr>
<tr>
<td>LENHDR</td>
<td>32</td>
<td>Length of header record 3</td>
<td>239</td>
</tr>
<tr>
<td>MPSIG</td>
<td>33</td>
<td>Position in header record 3 of first word of sigma</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value list. sigapb(1) = REALHD(MPSIG)</td>
<td></td>
</tr>
<tr>
<td>MPLAT</td>
<td>34</td>
<td>Position in header record 3 of list of latitude lines</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hdlat(1) = REALHD(MPLAT)</td>
<td></td>
</tr>
<tr>
<td>MPWTS</td>
<td>35</td>
<td>Position in header record 3 of Gaussian weights</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hdwt(1) = REALHD(MPWTS)</td>
<td></td>
</tr>
<tr>
<td>MPFLDS</td>
<td>36</td>
<td>Position of header field information list in header</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>record 1 (integer values)</td>
<td></td>
</tr>
<tr>
<td>MPCFLD</td>
<td>37</td>
<td>Position of field information list in header record</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 (character values), given as a number of 8-character</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>items from the start of record 2</td>
<td></td>
</tr>
<tr>
<td>MFLDS</td>
<td>38</td>
<td>Integer field information, dimensioned (3, nfldh).</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See description of MFLDS array in Table III.4 on page 130.</td>
<td></td>
</tr>
</tbody>
</table>
Table III.2: History File Header Record 2 — Character Variables*

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCASE</td>
<td>1</td>
<td>C*8</td>
<td>Case identifier</td>
<td>clsst01</td>
</tr>
<tr>
<td>MCSTIT</td>
<td>2-11</td>
<td>C*80</td>
<td>Case title</td>
<td>Test run</td>
</tr>
</tbody>
</table>

**Current History File**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNHSTC</td>
<td>12-21</td>
<td>C*80</td>
<td>Pathname of this history file</td>
<td>/user/csm/clsst01/ccm3/hist/h0001</td>
</tr>
<tr>
<td>LDHSTC</td>
<td>22</td>
<td>C*8</td>
<td>Date this header record was written (MM/DD/YY)</td>
<td>05/06/96</td>
</tr>
<tr>
<td>LTHSTC</td>
<td>23</td>
<td>C*8</td>
<td>Time this header record was written (HH:MM:SS)</td>
<td>16:14:20</td>
</tr>
<tr>
<td>LSHSTC</td>
<td>24</td>
<td>C*8</td>
<td>Sequence number of run producing this file</td>
<td>runmodel</td>
</tr>
</tbody>
</table>

**First History File of Case**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNHSTF</td>
<td>25-34</td>
<td>C*80</td>
<td>Pathname of first history file for this case</td>
<td>/user/csm/clsst01/ccm3/hist/h0001</td>
</tr>
<tr>
<td>LDHSTF</td>
<td>35</td>
<td>C*8</td>
<td>Date this case started (MM/DD/YY)</td>
<td>05/06/96</td>
</tr>
<tr>
<td>LTHSTF</td>
<td>36</td>
<td>C*8</td>
<td>Time this case started (HH:MM:SS)</td>
<td>16:04:02</td>
</tr>
<tr>
<td>LSHSTF</td>
<td>37</td>
<td>C*8</td>
<td>Sequence number of run that started this case</td>
<td>runmodel</td>
</tr>
</tbody>
</table>

**Initial Dataset File**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNHSTI</td>
<td>38-47</td>
<td>C*80</td>
<td>Pathname of initial dataset file</td>
<td>/OLSON/datasets/T42/CCM3SEP1</td>
</tr>
<tr>
<td>LDHSTI</td>
<td>48</td>
<td>C*8</td>
<td>Date initial dataset was created (MM/DD/YY)</td>
<td>12/28/95</td>
</tr>
<tr>
<td>LTHSTI</td>
<td>49</td>
<td>C*8</td>
<td>Time initial dataset created (HH:MM:SS)</td>
<td>18:32:15</td>
</tr>
</tbody>
</table>

*All character strings are left-justified, blank-filled, and multiples of 8 characters.
### Table III.2: History File Header Record 2 — Character Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSHSTI</td>
<td>50</td>
<td>C*8</td>
<td>Sequence number of run creating initial data</td>
<td>runmodel</td>
</tr>
<tr>
<td><strong>Time-Invariant Boundary Dataset File</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNHSTT</td>
<td>51-60</td>
<td>C*80</td>
<td>Pathname of time-invariant boundary dataset</td>
<td>tibds.T42.0596</td>
</tr>
<tr>
<td>LDHSTT</td>
<td>61</td>
<td>C*8</td>
<td>Date last header record was written on time-invariant boundary dataset (MM/DD/YY)</td>
<td>12/27/91</td>
</tr>
<tr>
<td>LTHSTT</td>
<td>62</td>
<td>C*8</td>
<td>Time last header record was written on time-invariant boundary dataset (HH:MM:SS)</td>
<td>09:12:22</td>
</tr>
<tr>
<td>LSHSTT</td>
<td>63</td>
<td>C*8</td>
<td>Sequence number of run writing time-invariant boundary dataset</td>
<td>- 1 -</td>
</tr>
<tr>
<td><strong>Time-variant SST Dataset File</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNHSTVS</td>
<td>64-73</td>
<td>C*80</td>
<td>Pathname of SST boundary dataset</td>
<td>srt-sst.T42.0596</td>
</tr>
<tr>
<td>LDHSTVS</td>
<td>74</td>
<td>C*8</td>
<td>Date SST boundary dataset was created (MM/DD/YY)</td>
<td>12/23/91</td>
</tr>
<tr>
<td>LTHSTVS</td>
<td>75</td>
<td>C*8</td>
<td>Time SST boundary dataset was created (HH:MM:SS)</td>
<td>15:01:43</td>
</tr>
<tr>
<td>LSHSTVS</td>
<td>76</td>
<td>C*8</td>
<td>Sequence number of run creating SST boundary data</td>
<td>- 1 -</td>
</tr>
</tbody>
</table>

*All character strings are left-justified, blank-filled, and multiples of 8 characters.*
### Table III.2: History File Header Record 2 — Character Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Word Number</th>
<th>Format</th>
<th>Description</th>
<th>T42 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNHSTVO</td>
<td>77-86</td>
<td>C*80</td>
<td>Pathname of ozone boundary dataset</td>
<td>ozn.T42.0596</td>
</tr>
<tr>
<td>LDHSTVO</td>
<td>87</td>
<td>C*8</td>
<td>Date ozone boundary dataset was created (MM/DD/YY)</td>
<td>09/21/92</td>
</tr>
<tr>
<td>LTHSTVO</td>
<td>88</td>
<td>C*8</td>
<td>Time ozone boundary dataset was created (HH:MM:SS)</td>
<td>13:34:26</td>
</tr>
<tr>
<td>LSHSTVO</td>
<td>89</td>
<td>C*8</td>
<td>Sequence number of run creating ozone boundary data</td>
<td>-1-</td>
</tr>
<tr>
<td>MCFLDS</td>
<td>90</td>
<td>C*8</td>
<td>Character field information, dimensioned (2,NFLDH). See description of MCFLDS array in Table III.4 on page 130.</td>
<td>--</td>
</tr>
</tbody>
</table>

*All character strings are left-justified, blank-filled, and multiples of 8 characters.

### Table III.3: History File Header Record 3 — Real Variables

<table>
<thead>
<tr>
<th>Variable Name*</th>
<th>Word Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALHD (MPSIG)</td>
<td></td>
<td>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</td>
</tr>
<tr>
<td>REALHD (MPLAT)</td>
<td></td>
<td>Points to the first word, relative to the start of the header, of a list of NOREC Gaussian latitudes in degrees (south to north)</td>
</tr>
<tr>
<td>REALHD (MPWTS)</td>
<td></td>
<td>Points to the first word of a list of NOREC Gaussian weights corresponding to the Gaussian latitudes</td>
</tr>
</tbody>
</table>

*MPSIG, MPLAT, and MPWTS are from header record 1, words 33, 34, and 35, respectively.
Of the variables shown in the header records, a few are of particular importance and require some additional explanation:

**LENHDI**  Word one of the first (integer) header record. Equals the exact length of the first header record in words, which varies depending on the length of the array **MFLDS** (starting at word 38 of the integer header). The 1-D array **comhdi** is set equal to **LENHDI** via a Fortran *equivalence* statement, in order to facilitate accessing variables in the integer header record.

**LENHDC**  Word 31 of the integer header record. Equals the exact length of the second (character) header record in 8-byte words. Varies depending on the length of the array **MCFLDS** (starting at word 90 of the character header).

**LENHDR**  Word 32 of the first header record. Equals the exact length of the third (real) header record in words. Varies depending on the length of the array **REALHD**.

**MCASE**  The first word of the character header record and the case identifier. The 1-D array **comhdc** is set equal to **MCASE** via a Fortran *equivalence* statement, in order to facilitate accessing variables in the character header record.

**MFTYP**  History file format identifier. Provides file format information for postprocessors.

**REALHD**  The real header record array. This 1-D array contains the data values from the following real arrays ordered consecutively: \( \text{sigapb}(2\times\text{plev}+1) \), \( \text{signa}(2\times\text{plev}+1) \), \( \text{sigb}(2\times\text{plev}+1) \), \( \text{hdlat}(\text{plat}) \), and \( \text{hdwt}(\text{plat}) \).

The 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 III.4 shows details of the contents of these two arrays.

### (c) Latitude Data Records

Following the header records in each time sample are **plat** latitude data records. All words in a latitude data record are floating point. If the Model is run multitasked, then the order in which the latitude records are written to a history file is indeterminate. 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.F**).

Figure III.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
Table III.4: Description of Field Information List Arrays, MFLDS and MCFLDS

<table>
<thead>
<tr>
<th>Variable Name†</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLDS(1, ifld)</td>
<td>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.</td>
</tr>
<tr>
<td>MFLDS(2, ifld)</td>
<td>Pointer to the first field value in packed data record.</td>
</tr>
<tr>
<td>MFLDS(3, ifld)</td>
<td>Data-packing flag (set to 1 if not packed).</td>
</tr>
<tr>
<td>MCFLDS(1, ifld)</td>
<td>Field name for field number ifld. (8 characters)</td>
</tr>
<tr>
<td>MCFLDS(2, ifld)</td>
<td>Units of this field (SI units). See Table IV.1 on page 155. (8 characters)</td>
</tr>
</tbody>
</table>

†The index ifld goes from 1 to nfldh (the number of fields in the history buffer).

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

- The first word of the record is a 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 \( p_{\text{lon}} \) longitude points, for the given latitude index. If a field is multi-level, then it contains grid-point data at \( p_{\text{lon}} \times p_{\text{lev}} \) points, for the given latitude index.
(d) History Buffer

The history buffer is dynamically allocated storage space that is used to accumulate history file data fields. As described in Sec. II.B, 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 it may be many iterations between writing time samples to a history file, it is necessary to have storage space available in which the history file fields can be accumulated and stored. Usage of namelist variables PRIMARY, EXCLUDE, and AUXF permit the user to define how many fields should be saved in the primary history file and what fields, if any, should be saved in the auxiliary history files. This information is used by the Model along with the default fields on the Master Field List (see Table II.9 on page 53) to determine the length of the history buffer, lhbuf, required to store all the history file fields.

Having calculated the length of the history buffer, the Model then dynamically declares memory to hold this buffer. Depending on the machine being used and the memory available, the history buffer can be implemented using out-of-core SSD memory, or retained entirely within in-core memory. The namelist variable INCOREHST is used to decide whether or not the history buffer should be kept in-core (INCOREHST = .true.) or out-of-core (INCOREHST = .false.).

As mentioned previously, time samples are written to history files from within a multitasked loop over latitudes. Thus, when a given processor writes to the disk history file, the fields output correspond to grid-point values along a single latitude line. It was convenient, therefore, to define the length of the history buffer to be the combined lengths of the history file data fields for each declared history file (primary and auxiliary) for a single latitude value, i.e. lhbuf equals (plon x number of single level fields in all history files) + (plon x pllev x number of multi-level fields in all history files). Recall plon is the number of longitude points and pllev is the number of vertical levels. The Model sorts through and organizes the fields in the history buffer according to its associated history file (primary or auxiliary).

Figure III.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. The arrays npplen and nrlen contain the packed and unpacked field lengths of each
Figure III.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. The lengths of the packed and unpacked latitude data records for the primary and first auxiliary history files are shown by \( nplen \) and \( nrlen \), respectively.

data record. Recall namelist variable \( NDENS(1) \) specifies the packing density of fields in each history file. For the buffer shown in Figure III.9, \( NDENS(1) = 2 \), thus \( nplen(1) \) (the packed length) is one-half the length of \( nrlen(1) \) (the unpacked length) as shown.

Allocation of memory for the history buffer occurs during the initialization phase of a run, in subroutine \texttt{inht}. If the buffer is contained entirely in-core, then the Model calls routine \texttt{getmem} to dynamically allocate a segment of in-core memory of length \( plat \times lhbuf \). 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-stepping phase of the Model, field values to be output in 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 \texttt{outfld}. Calls to \texttt{outfld} occur within a multitasked loop over latitudes, from routines \texttt{linemsbc} and \texttt{linemsac} and below. When running multitasked,
each process will perform calls to outfld.F and load the history buffer with fields from a single latitude index. In order to facilitate accessing the section of the in-core history buffer relevant to that latitude index, the pointer array hbuf is used. The following block of code, extracted from linemsbc.F, shows how this is accomplished.

```fortran
subroutine linemsbc(lat , tdt, ps , psml, 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 ',psml,pLt,ps,psml,u3,history

Example code of the in-core history buffer association

Allocation of the out-of-core history buffer

In this code, lat refers to the particular latitude index that is currently being calculated, and is passed as an argument to linemsbc.F from a multitasked loop in scanlbc.F. 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.F that is shown will load the surface pressure array, psml, into the history buffer. A similar call to outfld.F exists for every history file field.

If the history buffer is declared to be out-of-core then the Model will use the SSD (if available\(^1\)) to accumulate history file fields. As described in "Out-of-Core Data Storage: The SSD Work Units" on page 108, 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.F and linemsac.F), the following four steps are executed on each processor:

1) In-core memory of length lhbuf is allocated from the heap using routine getmem.F.

---

1. 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.
2) The portion of the history buffer that corresponds to the specific latitude index being processed is read in from the SSD using routine readric.F, 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.

3) History fields are accumulated in array hbuf through calls to routine outfld.F.

4) Prior to completion of the multitasked routine, hbuf is written to the out-of-core history buffer located on the SSD using routine writeric.F. The in-core space that was allocated for hbuf is then released by calling freemem.F.

Example code of the out-of-core history buffer allocation

The following block of code shows how the out-of-core history buffer is accessed in linemsbc.F. This differs from the previous block of code shown only in that it includes code to access the out-of-core history buffer from the SSD. Similar to the in-core procedure, the current latitude index

```
subroutine linemsbc(lat ,tdt ,ps ,psml ,u3 ,
.
.
pointer (phbuf,hbuf)
real hbuf(lhbuf) ! history buffer pointee array
.
.
if (incorhst) then
   phbuf = loc(hbufic(l,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 ',psml,plond,lat,hbuf)
.
.
if (.not.incorhst) then
   call writeric(sunit,hbuf,lhbuf,lat) ! write to SSD
   call freemem(phbuf)
endif
.
.
return
```

lat is passed as an argument to linemsbc.F from scan1bc.F. The pointee array hbuf and its pointer phbuf are next declared. Because
INCORHST is false when the history buffer is kept out-of-core, the Model calls getmem.F and readric.F 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.F, as shown for the surface pressure array psml. After all history fields have been loaded into the hbuf array, and prior to executing the return statement in linemscbc.F, the history buffer fields are written to the out-of-core history buffer on the SSD by calling routine writeric.F. Once this has been completed, freemem.F is called to release the in-core memory and return it to the heap.

(e) History File Code Flow

Many routines are involved in the initialization and writing of the history files. Because it can be difficult to track these routines within the code, a brief summary of the code flow pertaining to the history files is given below. To view the relative locations within the Model of the routines to be cited below, refer to the CCM3 calling tree in Appendix B.

Initialization of the history files occurs in several routines prior to the calling of the time-stepping routine stepon.F by ccm3.F. The namelist variables associated with the history file controls are initialized in routine preset.F and read into the Model in data.F. A Fortran unit number for an out-of-core SSD file is assigned in lunits.F. If conducting a restart or regeneration run, existing history files that are not full (contain less that MFILT(i) time samples) are opened and repositioned in restrt.F.

Routine intht.F 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.F to build the Master Field List (see Table 11.9 on page 53). Next, fldlst.F is called upon to construct the history buffer from the default fields in the Master Field List and as directed by namelist variables EXCLUDE, PRIMARY, and AUXF. Among the variables determined in fldlst.F are the following:

mtapes - the number of primary and auxiliary history files declared

lhbuf - length of the history buffer

hbufpt - pointers to the beginning of latitude data records for each declared history file in the history buffer

mflds and mcflds - field information lists; see Table III.4

npnt and nupnt - arrays of packed and unpacked field pointers (point into the history buffer)
nplen and nr1en - arrays of packed and unpacked field lengths of the latitude data records of each history file.

After defining the content of history buffer and files in fdlst.f, Fortran unit numbers are assigned for the disk history files in array hunit using function navu.f. Next, int hed.f is called to initialize the variables in the history file header common to all history files. Routine int hed.f then either dynamically allocates memory for the in-core history buffer, or opens the SSD file necessary for the out-of-core history buffer. Finally, if any history buffer regeneration files exist (as may happen in a restart or regeneration run) then splitg.f is called to read its contents into the in-core history buffer or the out-of-core SSD file.

The final history file routine called during the initialization phase of the Model is rgnhbf.f. This routine initializes the history buffer regeneration files (if needed).

Writing fields to the history buffer and files

Writing of the history fields to the buffer and writing the history buffer contents to a history file, occurs from within the time-stepping loop of stepon.f. History fields are accumulated in the history buffer on every timestep. However, as described in Sec. II.B, a time sample is generally not written to the disk history file on every timestep, but as directed by nam elist variable NHTFRQ(i). The Model sets logical variable hstwr(i) to true if it is time to write a time sample to a given history file, otherwise hstwr(i) is set false. On the first timestep of a run, hstwr(i) is set true, so that a time sample is always written to the a history file. After incrementing the timestep (nstep=nstep+1), function whenwr.f is invoked by stepon.f to determine if the history file should be written to on the current timestep (and hstwr(i) is accordingly set true or false). Of the subroutines called by stepon.f, only scan1bc.f, scanlac.f, and wrapup.f directly interact with the history buffer or files.

The driver routine for the first Gaussian latitude scan is scan1bc.f. This routine checks if a hstwr(i) is true, and if so, writes the history file header to the disk file. This routine will also open a new disk history file if required (that is, if the previous disk file has been filled with MFILT(i) time samples, then a new disk file is opened). Within a multitasked loop, scan1bc.f calls routine linemsbc.f. In this routine, many of the history fields are stored in the history buffer through calls to outfld.f. Also, if hstwr(i) is true, linemsbc.f will call writup.f, and write a latitude data record from the history buffer to the disk history file. Routine writup.f employs wshist.f, wrtarr.f, and restr.f to pack the data fields, write to the disk file, and reset the values within the history buffer to zero.

The driver routine for the second Gaussian latitude scan is scanlac.f. This routine calls linemsac.f from within a multitasked loop. Routine
linemsac.F, in turn, stores many fields in the history buffer via calls to outfld.F.

Subroutines wrapup.F closes any full history files and disposes them to the NCAR Mass Storage System if requested (and if on a Cray machine that is linked to the MSS). Routines invoked by wrapup.F are bldcom.F, savdis.F, and wsds.F.

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. Refer to these header files for a detailed description of the common blocks.

2. Initial and Boundary Condition Datasets

The contents and usage of the initial and boundary datasets were described in detail in “Model Input Datasets” on page 45. Included in this description was a list of the fields included in each dataset as well as an indication of which routines are responsible for reading and processing the datasets. As discussed in “Model Input Variables - Namelist Input” on page 19, the names of the initial datasets read by the Model are specified in the namelist input (both namelists CCMEXP and LSMEXP). It was also mentioned that the input datasets are written in history file format (excluding the LSM input datasets FSURDAT and FINIDAT, which are written as ASCII text files). Thus, the history file format previously defined (a three-record header followed by plat latitude data records) is used as the format for the input datasets NCDATA, BNDTI, BNDTVS, and BNDTVO. Refer back to Figure III.7 on page 122 for an illustration of the structure of a history file.

The header records required on an input dataset contain exactly the same fields as those shown in Table III.1 to Table III.3 for a history file header. This is because the same routines, rdharr.F and mkslic.F, are used to read the header records and latitude data records of both the input datasets and the history files. Consequently, even though all the fields shown in these tables are required to be in the header, only a subset of these fields are actually needed to process an input dataset. A list of this subset header fields that must be set in order for the Model to run successfully are shown in Table III.5. The header on an input dataset not only provides information for use by post-processors and archives the Model run (as done for a history file), it also provides some information necessary to start a run.

3. Restart/Regeneration Files

As explained in “Restart and Regeneration Datasets” on page 63, the restart and regeneration files contain information required to restart a run from its termination point, to regenerate corrupt history files, or to begin a branch run. Regeneration files are automatically written if namelist variable NSVSN is set. One restart and four different types of regeneration files may be written during a run. The restart file along with the master and pri-
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Record Number</th>
<th>Word Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENHDI</td>
<td>1</td>
<td>1</td>
<td>Length of header record 1 (integer values)*</td>
</tr>
<tr>
<td>NRBD</td>
<td>1</td>
<td>5</td>
<td>Number of records before data records</td>
</tr>
<tr>
<td>NLON</td>
<td>1</td>
<td>9</td>
<td>Number of longitude points in one latitude line*</td>
</tr>
<tr>
<td>NLONW</td>
<td>1</td>
<td>10</td>
<td>Number of longitude points written on this dataset (must be equal to or greater than NLON)</td>
</tr>
<tr>
<td>NOREC</td>
<td>1</td>
<td>11</td>
<td>Number of latitude lines*</td>
</tr>
<tr>
<td>NLEV</td>
<td>1</td>
<td>12</td>
<td>Number of vertical levels*</td>
</tr>
<tr>
<td>NTRM</td>
<td>1</td>
<td>13</td>
<td>M spectral truncation parameter*</td>
</tr>
<tr>
<td>NTRN</td>
<td>1</td>
<td>14</td>
<td>N spectral truncation parameter*</td>
</tr>
<tr>
<td>NTRK</td>
<td>1</td>
<td>15</td>
<td>K spectral truncation parameter*</td>
</tr>
<tr>
<td>NFLDH</td>
<td>1</td>
<td>16</td>
<td>Number of fields on this dataset</td>
</tr>
<tr>
<td>NDCUR</td>
<td>1</td>
<td>22</td>
<td>Current day number**</td>
</tr>
<tr>
<td>NSCUR</td>
<td>1</td>
<td>23</td>
<td>Seconds of the current day**</td>
</tr>
<tr>
<td>NCDATE</td>
<td>1</td>
<td>26</td>
<td>Current date**</td>
</tr>
<tr>
<td>NCSEC</td>
<td>1</td>
<td>27</td>
<td>Seconds of the current date**</td>
</tr>
<tr>
<td>LENHDC</td>
<td>1</td>
<td>31</td>
<td>Length of header record 2 (character values)</td>
</tr>
<tr>
<td>LENHDR</td>
<td>1</td>
<td>32</td>
<td>Length of header record 3 (real values)</td>
</tr>
<tr>
<td>MPFLDS</td>
<td>1</td>
<td>36</td>
<td>Pointer to integer header field list information</td>
</tr>
<tr>
<td>MPCFLD</td>
<td>1</td>
<td>37</td>
<td>Pointer to character header field list information</td>
</tr>
<tr>
<td>MFLDS</td>
<td>1</td>
<td>38</td>
<td>Integer field list information</td>
</tr>
<tr>
<td>MCFLDS</td>
<td>2</td>
<td>77</td>
<td>Character field list information</td>
</tr>
</tbody>
</table>

* These values are used only to check against Model parameter settings. They are later overwritten in the history file header before writing.

** These four values provide the starting time information for the model run. They may alternatively be input using namelist parameters NNDBAS, NNSBAS, NNBDAT, and NNBSEC.
mary regeneration files are always written (if NSVSN is set), while secondary and history buffer regeneration files are only written if necessary. The frequency at which regeneration files are written is determined by namelist variable NREFRQ (unless NINAVG (1) = 'Q', in which case MRESFQ is used in place of NREFRQ). The maximum size of a single regeneration file is MXSZRG (defaults to 380 Mb). If a given regeneration file is larger than this, it will be split over two or more physical disk files, as described in Sec. II.E (the primary, secondary, and history buffer regeneration files can be very large). The maximum number of disk files a regeneration file may be split over is limited by the Fortran parameters prng1, prng2, and prng3 (set in header file comlun. h; equal to five at the time of release of CCM3). The NCAR Mass Store system can be used to archive restart and regeneration files on tape. The regeneration files are written as sequential access binary files. These files are intended only to be written to or read by the Model, so no facility exists with which to conveniently view these files.

In the text that follows, a general description of the content of the restart and regeneration files is provided. A brief outline of the code flow associated with these files is also discussed. To determine the exact contents of a particular regeneration file, refer to the specific routine responsible for writing the file (indicated below).

(a) Restart and Regeneration File Contents

The restart file (namelist variable NSVSN) is merely an ASCII text file containing the Mass Store pathname of the most recently written master regeneration dataset. Thus, for a "restart run" (namelist variable NSREST=1), the Model will read the restart file, acquire the appropriate regeneration files, and commence running from the end point of the previous run. The Model first checks the current directory for the restart file, then searches the MSS. The restart file is written from routine wrapup. F.

The master regeneration file contains information necessary to restart the Model. Variable values 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 regeneration file specifications. The master regeneration file is written from routine wsds . F, which is called by wrapup. F.

The primary regeneration file contains data from the main Model buffers and 3-dimensional arrays, allowing the processing of a continuation run to set all variables as they were at the time the restart/regeneration datasets were written. Data from the following common blocks are saved on the primary regeneration file: /coreioab/, /com3d/, /comslt/, and /comsrf/ (see the header files of the same name for details of these common blocks). For the standard Model configuration at T42 resolution, all of this data will fit in a single file of approximately 36 Mb. However, for
higher-resolution runs, this may not be true. The Model will calculate the expected length of the dataset and compare it with MXSZRG. If larger than MXSZRG, the data will be split up by latitudes, and written to separate files. The primary regeneration file is written from routine wrtrs1.F, which is called by scanlbc.F.

The secondary regeneration file is required only if the primary history file write frequency (NHTFRQ(1)) does not evenly divide the absorptivity/emissivity calculation interval IRADAE (see Table II.1 on page 20). In this case, absorptivities/emissivities would not necessarily be calculated on the first timestep after restart, and therefore must be saved on a regeneration dataset. 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 regeneration file over latitudes if necessary. This file is written from routine wrtrs1.F.

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

(b) Restart and Regeneration Code Flow

Several routines are involved in the initialization, reading, and writing of the regeneration files. A brief summary of the code flow pertaining to the regeneration files is given below. To see the relative locations within the Model of the routines to be cited below, refer to the CCM3 calling tree in Appendix B.

Initialization of the regeneration files occurs in several routines prior to the calling of stepon.F by ccm3.F. The namelist variables associated with the regeneration file controls are initialized in preset.F and read into the Model in data.F. Fortran unit numbers and filename extensions are assigned for the primary and secondary regeneration files in routine lunits.F. If performing an initialization run, routine initial.F is called, in which the size specifications of the primary and secondary regeneration files are calculated, including whether or not it is necessary to split these files over more than one disk file (or Mass Store tape).

If conducting a continuation run (restart, regeneration, or branch), ccm3.F will call routine resume.F instead of initial.F. This routine is the driver that reads in the appropriate datasets and performs the necessary initializations. Routine restrt.F, called by resume.F, does most of the work of reading in the regeneration files. If a restart run, then routine restrt.F first opens and reads the restart file. From this file is obtained
the path and filename of the master regeneration file. If a regeneration or branch run, then the path and filename of the master regeneration file are obtained from the namelist. Next, readrg.F and splitf.F are called to read the master and primary regeneration files, respectively. If secondary regeneration files exist, they are then read in by restrt.F. After positioning any partially full history files for additional data, restrt.F returns control to resume.F and eventually to ccm3.F.

Routine intht.F is the driver that initializes new history files and determines the contents and size of the history buffer. This routine also assigns Fortran unit numbers and filename extensions to the history buffer regeneration files. If conducting a restart or regeneration run (but not a branch run) and if history buffer regeneration files exist, then intht.F will call routine splitg.F to read in these files and load their contents into the history buffer.

The last routine called during the initialization phase of the Model is rgnhbf.F. This routine initializes the history buffer regeneration files (if needed).

Writing of the restart and regeneration files occurs from within the time-stepping loop of stepon.F. As described in Sec. II.B, the regeneration files are not written to on every timestep, but as indicated by namelist variable NREFRQ (or MREFRQ if NHTFRQ(1) = 'Q'). The logical variable rstwr is used to determine if it is time to write the regeneration files. No regeneration data is written on timestep zero. After incrementing the timestep (nstep=nstep+1), a test is performed using NREFRQ (or MREFRQ) to determine if the restart and regeneration files should be written during the new timestep (and rstwr is accordingly set true or false). Of the subroutines called by stepon.F, only scanlbc.F and wrapup.F are directly involved in writing the restart and regeneration files.

The driver routine for the first Gaussian latitude scan is scanlbc.F. This routine checks if the variable rstwr is true, and if so, calls routines rgnfis.F and wrtrsl.F. In rgnfis.F, the names of the master and primary regeneration files are assigned and the files are opened. Next, routine wrtrsl.F writes data into the primary regeneration file (writing the master regeneration file is deferred until wrapup.F is called). In addition to writing the primary regeneration file, wrtrsl.F also determines if secondary and history buffer regeneration files are to be written on the current timestep. If so, the appropriate files are opened and written.

Routine wrapup.F completes the task of writing the regeneration files begun in scanlbc.F. First the master regeneration file opened in rgnfis.F is written to by calling routine wsds.F. All regeneration files are then closed, and disposed to the NCAR Mass Storage System (if applicable) via routine savdis.F. Once this has been completed, wrapup.F writes the restart file, then calls savdis.F to close and dispose the file.
A few common blocks are associated with the regeneration files. These are defined in the header files `comlun.h` and `comloc.h`. Refer to these header files for a detailed description of the common blocks.
F. CCM3 Disk and Mass Store File Management

File management in the Model includes all activities related to the creation, use and maintenance of sequential and direct-access files. Emphasis is placed on mechanisms for maximizing storage utilization and minimizing file processing time. Specifically this includes acquiring files needed as input from the NCAR Mass Storage System (MSS), accessing data from these files as they reside on local disk, using standard Fortran I/O of various kinds, creating output files on disk, and disposing them to the MSS. In this discussion, we will describe how CCM3 provides control over these operations.

Several input datasets are required to begin a run or continue a previous run. All these are described in detail in “Model Input Datasets” on page 45 and “Restart and Regeneration Datasets” on page 63. Since the input datasets must exist on a local file system, in order to locate them the Model will:

- first search for a dataset in the current directory,
- then in a user specified directory,
- and as a last resort will read the dataset off of the MSS.

Filepaths passed via the namelist variables NCDATA, BNDTI, BNDTVS, BNDTVO, FSURDAT, and FINIDAT will be used.

Typically, the input datasets indicated by CCMEXP namelist variables contain a complete MSS filepath and will look something like /XXX/yyyy/zzz/filename. As a first attempt at finding the dataset, the Model will strip away the MSS part of the filepath and check if filename exists in the current directory. If the file is not found, the Model prepends a user specified directory (namelist variable LCROOT) to the MSS filepath and checks again for the dataset. The default for LCROOT is /ccm. Therefore, if LCROOT is not otherwise specified, then the Model would look in directory /ccm/XXX/yyyy/zzz/filename. If the file has not been found in either of the above locations, the Model will make a system call to retrieve the file from the MSS. The LSM input datasets are searched for only in the directory specified by the LSMEXP namelist variables (i.e. the Mass Storage System is not searched).

All the input datasets except the LSM datasets are in history file format and are formatted as COS blocked files on the Cray PVP and Cray T3D architectures and as 64-bit IEEE floating-point format on the SUN and IBM RS6K architectures. Both COS-blocked and IEEE-blocked datasets are available, and can be obtained at the same ftp site as the CCM3 code (see...
"How to Build and Run CCM3" on page 9). LSM datasets are ASCII text files.

As the Model runs it creates and disposes several output datasets to the MSS, if on an NCAR machine linked to the MSS. These files are described in detail in "Model Output Datasets" on page 51. All history and regeneration files are created in the local directory. Depending on the resolution, the frequency at which history and regeneration files are output, and the size of the file system upon which the Model is running, the aggregate amount of output data created could easily overflow local disk space. Three control mechanisms have been adopted to help alleviate this problem: allowing the user to control (1) the rate at which data is generated; (2) the size of the output data; and (3) disposing the data to an off-line device. All of this functionality is provided via the namelist mechanism and is described in "Model Input Variables - Namelist Input" on page 19. Those namelist variables which are important to disk and file management and which the user should fully understand are highlighted below.

Control over the rate at which history and restart data is generated is provided through the namelist variables NHTFRQ, NREFRQ, NSVSN, and NINAVG. By default a set of regeneration files will be written every time a history file is filled and disposed (NHTFRQ (1) X MFLIT (1) timesteps). Setting NREFRQ to a number greater than one allows the output of regeneration data after multiple history files have been written. Setting NSVSN to blanks (as done by default) results in no regeneration data being written at all. Use of this option is not recommended since it prevents the user from continuing the run once it has stopped. Setting NINAVG (1) to "Q" provides the user with the option of producing history information averaged over each simulated month. If this option is in effect, the Model will ignore the values of NHTFRQ (1), NREFRQ, and MFLIT (1) and use the value of MRESFQ (refer to Table II.1 on page 20).

Control over the size of the output data is achieved by limiting the number of fields and thus the amount of data written to the history file, and by compressing the data before it is written to local disk. The namelist variables EXCLUDE and NDENS provide the aforementioned functionality. EXCLUDE provides a run time mechanism for removing default fields from a history file while NDENS gives the user the ability to compress the history file before they are written to disk (NDENS is only available on an NCAR Cray).

Once an output file is written to the local disk, it may be disposed to an MSS tape. The MSS retention time namelist variables IRT and RIRT provide the option of saving history and regeneration files to the NCAR MSS for the specified number of days. By default IRT and RIRT are set to 365 days. Thus, if running CCM3 on an NCAR Cray, once output datasets are written locally, they will be disposed to MSS and removed from the local disk. This is true of all output datasets, with the exception of the final set of

CCM3 Disk and Mass Store File Management
history and regeneration data. Although this data will be sent to the MSS it is not deleted from the local directory to provide the option of continuing the run without the additional stage-up from the Mass Store to the local disk. If not running on an NCAR Cray, or if IRT and RIRT are set to zero, then the output datasets will be retained locally and no attempt will be made to dispose them to the MSS.

Several Cray-specific system routines are employed in order to execute system commands while CCM3 is executing. These system-specific calls are isolated to a few areas within the code, making the Model more easily portable to other computers. A summary of the system routines used in CCM3 are shown in Table III.6, along with a listing of the routines from which they are invoked.

**Table III.6: System Routine Calls in CCM3**

<table>
<thead>
<tr>
<th>System Routine</th>
<th>Description</th>
<th>Model Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local File Interface</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>setf</strong></td>
<td>Preallocate space on disk for local file</td>
<td>prealc.F</td>
</tr>
<tr>
<td><strong>Mass Store Interface</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>msread</strong></td>
<td>Read file from Mass Store</td>
<td>attach.F</td>
</tr>
<tr>
<td><strong>mserror</strong></td>
<td>Print Mass Store errors</td>
<td>attach.F, savdis.F</td>
</tr>
<tr>
<td><strong>mswrite</strong></td>
<td>Write file to Mass Store</td>
<td>savdis.F</td>
</tr>
<tr>
<td><strong>Interface with Run Script</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>getenv</strong></td>
<td>Get value of environment variable</td>
<td>data.F, igtseq.F</td>
</tr>
<tr>
<td><strong>Miscellaneous System Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>abort</strong></td>
<td>Fatal error stop, issue traceback</td>
<td>blowup.F, endrun.F</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td>Return current date as character string</td>
<td>ccm3.F, inthed.F, wrthdr.F</td>
</tr>
<tr>
<td><strong>clock</strong></td>
<td>Return current time as character string</td>
<td>ccm3.F, inthed.F, wrthdr.F</td>
</tr>
<tr>
<td><strong>ishell</strong></td>
<td>Issue system (shell) command</td>
<td>iostop.F (for error processing), prealc.F (for disk space preallocation)</td>
</tr>
</tbody>
</table>
CCM3 I/O is discussed in detail in “Out-of-Core Data Storage: The SSD Work Units” on page 108. All I/O is implemented using standard Fortran I/O statements. Although the work files used in the out-of-core implementation are unblocked to enhance I/O performance, the history file is written as a sequential, COS blocked dataset (due to the existence of related external programs, such as the CCM Processor which reads the history file). The regeneration datasets are also COS blocked. The secondary regeneration dataset, containing absorptivities and emissivities, is unblocked.

1. CCM3 Disk File Management on an NCAR Cray

As on many supercomputer systems, disk space on the Cray Y-MP at NCAR is a precious commodity. To enhance usability of the temporary file system, several logical divisions exist in the temporary disk space. The Model interacts with two of these—the job temporary space known as $\text{TMPDIR}$ and the user temporary space allocated under the directory `/usr/tmp`.

The directory pointed to by system environment variable $\text{TMPDIR}$ actually resides in `/usr/tmp`. It is created as a unique directory at login time. For batch jobs, it is created when the queue management subsystem NQS starts the job on the Cray. This directory exists only until the batch run script terminates, at which time it disappears along with all of its files.

The user temporary disk space, `/usr/tmp/$\text{LOGNAME}`, remain after job completion, but is subject to being “scrubbed” of individual files, based presumably on size and age of the file. At times, this translates into a retention time of only a few minutes! By making a $\text{TMPDIR}$ directory the “current directory” for the Model run, we assure that all local files are protected from scrubbing.

Most of the logical unit numbers in CCM3 are determined at execution time. Only the SDS work units are assigned in the run script. The Model keeps a catalog of logical unit numbers in use and when possible returns an unused number. After all units are assigned, a table of unit numbers and their usage is printed in the Model output.

A heavily used file system such as the temporary disk system on the Cray can become very fragmented. Large output files such as the history file are slow to read when this happens. For this reason, the Model preallocates space equal to the expected length of the output file at assign time using the `setf` command with the `-c` option. Subroutine `prealc.F` calls `setf` to find as much contiguous space as possible for an output file. This method is used in assigning all large output datasets, including the history files, and the primary and secondary regeneration files.

The Model is able to make asynchronous writes to the MSS by calling the system routine `mswrite` with the `nowait` option. This allows time integration to continue without having to wait for the file to reach the Mass Store.
The `mswrite` command will establish a link (`ln`) for the output file into a protected directory owned by the system before it returns control to the calling program. These protected links preserve the files until they are actually written to the Mass Store. The Model is then free to remove the local copy of any file which is written to the MSS as soon as the asynchronous call returns.

### 2. Use of the NCAR Mass Storage System

The Model uses the NCAR Mass Store System for archival storage of all output datasets. Input datasets are copied to the Cray disk from the MSS and accessed via Fortran-callable `msread`. The output history files and restart and regeneration datasets are disposed to the MSS via Fortran-callable `mswrite`. Routines that perform mass store operations are isolated in the Model to facilitate replacement when the Model is moved to another system.

The `msread` requests are issued as synchronous operations. Hence, the Model will wait until a read is completed before resuming execution. However, `mswrite` commands are always issued asynchronously because output files are written to the MSS only after they are completed. Thus, each output file will have a different name and there is no danger of overwriting a file before it actually gets to the Mass Store. No provision exists for read passwords on Model output tapes.

Output file naming in CCM3 is entirely automatic as described in “Naming the Primary and Auxiliary History Files” on page 62 and “Naming of Restart and Regeneration Files” on page 64. On an NCAR Cray, the output files of a batch run are written to disk in the directory pointed to by environment variable `$TMPDIR` and the namelist variable `CASEID`. This file will be removed once `mswrite` returns control to the Model. This prevents `$TMPDIR` from becoming full on long runs.

As was described in “Mass Store Archiving of History and Restart/Regeneration Files,” on page 67, the MSS directories in which the tapes are stored is given by:

- `/USERNAME/csm/CASEID/ccm3/hist` for history tapes.
- `/USERNAME/csm/CASEID/ccm3/rest` for restart and regeneration tapes.

### 3. Issuing Shell Commands from the Model

File operations that cannot be handled with Fortran library calls are accomplished by issuing shell commands via system routine `ishell`. The number of `ishell` calls in CCM3 is minimized by making use of the Fortran interface to commands such as `mswrite`. However, as mentioned above, it is advantageous to preallocate disk space for large files such as history...
and regeneration volumes. The only way to do this reliably is with the system `setf` command, which has no Fortran interface.
IV. 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 resolution, and adding a parameterization. This section provides some guidelines for making these kinds of changes in CCM3.

A. Modifying the Code

As described in “Overview of Running CCM3” on page 3, the CCM3 code is distributed in “make” form, with each subroutine and common block stored as a separate file. The UNIX make utility processes the Makefile in order to build the executable version of the Model. The Makefile instructs make to read the Objects and Depends files, and then directs the Fortran compiler to update any object (“.o”) files that are out-of-date or to create any that do not yet exist. An object file is out-of-date if modifications have been made to its source code (“.F” file) or any of the header (“.h”) files upon which a source code file depends (due to a #include statement within the source code). If a user has made changes to any of the existing source code or header files, then the executable file must be updated. This is accomplished very simply by re-running make.

If the user goes beyond making changes to existing code, and actually adds new routines or header files, then the Objects and Depends files used by the Makefile must be updated to reflect these changes. Included with the Model distribution in the ccm3/ directory is the PERL (Practical Extraction and Report Language) script makdep. This script builds the Objects and Depends files that the Makefile uses in creating/updating the CCM3 executable. To use makdep, type perl makdep on the command line in the ccm3/ directory.

To enhance portability and readability of the code, the source code contains UNIX cpp (C preprocessor) commands of the form #include, #if defined, etc. An example taken from routine writup.F is shown below:

```fortran
  subroutine writup(lat ,hbuf)
    ... model code ...

    #include<comhst.h>
    ... model code ...

    #if ( defined CRAY ) || ( defined T3D )
    ... model code ...
    #else
    ... model code ...
    #endif

    ... model code ...
```
This structure allows the implementation of distinct blocks of platform specific code within a single file. When *make* is run, *cpp* uses *if defined* logic to include or exclude blocks of code, based on which *cpp* tokens (e.g. CRAY, T3D, etc.) the user has defined. The *cpp* tokens are set in the file *params.h*, which is built by the *setup* script for a given platform, resolution, etc.

Several *cpp* tokens are used in CCM3:

- **CRAY, T3D, SUN, SGI and RS6K** delineate blocks of code which are specific to the respective machine architectures. These 5 options are mutually exclusive, therefore only one of them should be defined when building the code. The SGI option is for testing purposes and has not yet been validated, so it is currently not a supported option for CCM3. Also, some routines have two complete blocks of code for CRAY vs. non-CRAY. This is due to the fact that spectral coefficients are stored consecutively along diagonals of M-N wavenumber space when the target architecture is CRAY (optimal for vectorization), and along total wavenumber N otherwise (optimal for cache based architectures). Refer to “Spectral Data Structures” on page 112 for details about the storage of the spectral coefficients.

- **COUP_SOM** marks code necessary for calling the Slab Ocean Model (SOM), described in Sec. II.G on page 91.

- **COUP_OCE** configures the code for coupling to the Climate System Model (CSM). The message-passing interface to the CSM driver is documented in *The NCAR CSM Flux Coupler*, (Kauffman, 1996). This option is not compatible with the COUP_SOM or SPMD options.

- **SPMD** configures the code for message-passing on parallel platforms such as the T3D, SUN or RS6K, but not conventional CRAY (PVP) machines. When SPMD is specified, the COUP_OCE and CRAY options must not be defined.

- **DIAGNS** enables diagnostic printing in the convection routines.

- **PVME** is for use with IBM’s proprietary PVM implementation.

- **SHMEM** is T3D-specific and makes use of fast, low-level message passing routines rather than the generic (slower) PVM calls.

- **FORTFFT** must be defined when the Model is run using the Fortran version of the ECMWF FFT package. This *cpp* token provides the additional workspace needed by the fortran FFT code.

In addition to the above *cpp* tokens which are either defined or undefined, the C preprocessor is also used to perform textual substitution for resolution specific parameters in CCM3 and in the Land Surface Model. All res-
olution dependent substitutions are defined in the header files params.h (CCM3 header file) and preproc.h (LSM header file). The format of the cpp variables follows standard cpp protocol in that they are all uppercase versions of the Fortran variables which they define. Thus, the user might see a statement like

\[
\text{parameter}(\text{plat} = \text{PLAT})
\]

in a parameters header file. Since the C preprocessor is case-sensitive, when invoked it will replace the uppercase variable with what was defined in the CCM3 header file params.h. The processed line would then look like the following for the standard T42 resolution.

\[
\text{parameter}(\text{plat} = 64)
\]

The following cpp variables are set in the CCM3 header file params.h:

- **PCNST** is used to define the number of constituents which are carried in the Model. This parameter affects the size of the in-core data structure holding the prognostic variables.
- **PLON** defines the number of longitudes on the CCM3 transform grid.
- **PLAT** defines the number of Gaussian latitudes on the CCM3 transform grid.
- **PLEV** defines the number of vertical levels in the model.
- **PLEVR** defines the number of vertical levels over which radiation calculations are performed. The use of a separate vertical coordinate for the radiation calculations has not been tested. Therefore this parameter must currently be set to the same value as PLEV.
- **PTRM** defines the spectral truncation of the zonal wavenumber m.
- **PTRN** defines the spectral truncation of the total wavenumber n for zonal wavenumber 0.
- **PTRK** defines the maximum total wavenumber k, for any zonal wavenumber m.

The following cpp variables are set in the LSM header file preproc.h:

- **LSMLON** defines the number of longitudes for the LSM grid. Currently this parameter must be the same as the CCM3 longitude parameter PLON.
- **LSMLAT** defines the number of latitudes for the LSM grid. Currently this parameter must be equal to the CCM3 latitude parameter PLAT.
Changing the Model

- **LPT** defines the total number of land points for the LSM grid.
- **KPT** defines the dimension of LSM internal array kpt. This array is an expansion of the lpt array according to the surface types defined at each land point. Currently, the LSM recognizes up to five different surface types per land point.
- **NUMLV** defines the number of vectors which the kpt array is partitioned for shared-memory parallel processing.
- **COUP_CCM** is set whenever the LSM is being run within the CCM3. For the standard CCM3 run, this token is always defined.

### B. Adding New Variables

The most frequently performed changes to the Model in terms of adding new variables are: adding variables to a history file; adding new diagnostic fields; adding transported constituent; and adding variables within a physical parameterization. Of these changes, the former two will be discussed in this section, while the latter two will be discussed in Sec. IV.C and Sec. IV.E, respectively.

**Case 1: The variable is already known by the Model and the user wants to save it on a history file.**

If the field is in the Master Field List (see Table II.9 on page 53), the user needs to modify the code by un-commenting the outfild.f call for that field (remove the "C" from column 1). Next, the field is included in a history file by using either `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 II.1 on page 20.

If the field is not on the Master Field List, the user must add it to the list by modifying bldfld.f. Four pieces of information must be specified when a field is added to the Master Field List:

1) 8-character field name, left-justified, alphanumeric or spaces only.
2) 8-character units description. See Table IV.1 for a description of the convention used for CCM3 data fields units (the same as used in CCM2).
3) Level indicator:
   - `nsingle` = single-level
   - `nmultl` = multi-level at layer interfaces
   - `nmultf` = multi-level at layer mid-points
4) 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 IV.1: SI Units of History File Fields

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>meter</td>
</tr>
<tr>
<td>KG</td>
<td>kilogram</td>
</tr>
<tr>
<td>S</td>
<td>second</td>
</tr>
<tr>
<td>K</td>
<td>degree Kelvin</td>
</tr>
<tr>
<td>MOL</td>
<td>mole</td>
</tr>
<tr>
<td>N</td>
<td>Newton</td>
</tr>
<tr>
<td>PA</td>
<td>Pascal</td>
</tr>
<tr>
<td>J</td>
<td>Joule</td>
</tr>
<tr>
<td>W</td>
<td>Watt</td>
</tr>
</tbody>
</table>

Within bldfld.F, fields are added to the Master Field List by calling routine addfld.F. The four pieces of information previously discussed are passed to addfld.F in an argument list. An example extracted from bldfld.F is shown below:

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

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.F are the 8-character field name, the 8-character field units, nf (a field counter used by bldfld.F), the level indicator, and the active/inactive flag.

After adding the field to the Master Field List, space for the field must be allocated in the history file buffer. This is accomplished by incrementing parameter pfllds in header file pagrid.h. The user must then add an outfld.F call for the field at the proper location in the code. This means that outfld.F must be called either from linemsbc.F, linemsac.F, or below in the calling tree. For example, the outfld.F call for T, taken from linemsbc.F, is shown below:

```
call outfld('T ', t3mlplond, lat, hbuf)
```
The arguments in the call to `outfld.F` are as follows: the 8-character field name; the variable array in which it is stored; the first dimension of the data array (typically equal to the number of longitude points along the current latitude line being processed: either `plond` or `plon`); the latitude index; and the name of the history buffer, `hbuf`. In adding a call to `outfld.F`, 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 be added to a history file using `namelist` variables `PRIMARY` or `AUXF`, or removed via `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 buffer requires adding buffer field pointers in `common /comgrd/` and setting those pointers in subroutine `points.F`. If saved in the Model buffer then `parameters` in `pagrid.h` that determine buffer lengths must also be changed. Also, if used in routines `linemsbc.F` or `linemsac.F` or below and if saved in the Model buffer, then the variable must be passed as an argument into `linems[ba]c.F`). These `parameters` are illustrated in Figure III.2 on page 107 (`pbflnb`, `pbflna`, `plencl`, and/or `PCNST`). Follow directions under Case 1 to include the new variable in a history file.

C. Adding Constituents to the Model: The Semi-Lagrangian Transport

The `parameter` `PCNST` represents the number of constituents transported in the atmosphere by the Model. CCM3 carries one constituent (water vapor) by default, but has the capability of transporting up to 98 additional constituents depending on available in-core memory. To add one or more constituents the user must make two modifications. First, the `cpp` token `PCNST` (in `params.h`) must be increased by the number of constituents added. Second, the user must either create an initial dataset containing values for the new constituents, specify that they all be initialized to zero, or introduce code into subroutine `inidat.F` that sets the constituent values before the initial write to the work files (buffers).

The Model will then:

- Create array space for the new constituents and associated diagnostics.
- Advect all constituents using SLT.
- Apply vertical diffusion, convective transport and time-filtering to the new 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).

It is important to realize that any new constituents act strictly as passive tracers. There are no feedback mechanisms built into the Model that allow a constituent to affect the climate prediction. However, physical processes within the atmosphere (in addition to the vertical diffusion, etc., mentioned above) can be included to affect the transport of a constituent. Constituents are input and output to/from the Model as dry mass mixing ratio. Within routines aphys.f, tphysbc.f, and tphysac.f the units are converted to wet mass mixing ratio. Typical examples of processes that a user may include to influence a new constituent are listed below:

- 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} (Kg/Kg_{air}) for water vapor, 0.0 (Kg/Kg_{air}) for all others).

Since these user changes are fairly common, they are described in detail below, along with how to initialize the constituent data fields.

1. Initializing New Constituent Data Fields

As stated previously, if a new constituent is added, either the initial dataset (as indicated by namelist variable NCDATA) must be altered to include initial values of the constituent, or the initial values must be specified as zero, or code to initialize these values must be added. As described in Table II.1, namelist variable SETTRACE defaults to 'init', implying that the initial dataset will be read in order to initialize the constituent fields. Alternatively, constituent values can be initialized to zero if SETTRACE is defined as 'zero'. While specifying the initial values as zero using SETTRACE is a simple task, modifying the initial dataset or adding code to the Model requires additional development. Both of these techniques are described below.

If initial values of a new constituent are to be input, then the user must supply them to the initial dataset. This is accomplished through use of the CCM Processor (Buja, 1992), which has the capability of creating initial datasets for CCM3. Two conditions must be met by the initial data for the new constituent: the field name of the new constituent must match the field name automatically assigned by the Model, and initial values must be supplied for every grid point at the given resolution.
Changing the Model

Naming new constituents

The field name of a constituent is assigned in routine initcom.F, and saved in array tracnam(PCNST) (i.e. tracer name). Array tracnam, as well other 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 the letters “TR” (for tracer) 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 constituents have been added, then their assigned field names would be TR02 and TR03. The field names on the initial dataset should match these names.

Once the initial condition dataset has been created, the Model will automatically read the new constituent data into the proper array space (no new array space needs to be defined by the user). Output of the tracer field data to the primary history file is done by default. Refer to Table 11.9 on page 53 to see what other tracer-related fields are part of the Master Field List. Note, in the first row of this table the tracer name is represented by TRxx, where the “xx” stands for the two-digit number portion of the tracer name (e.g. 02, 03, etc.). A similar convention for terminating field names with an “xx” exists for all other tracer-related fields shown in Table 11.9.

Coding initial values of the constituents

Instead of adding constituent fields to the initial dataset, the user may add new code to routine inidat.F that set the initial values as desired. The procedure to add such code is as follows. Set namelist variable SETTRACE to 'init'. Within subroutine inidat.F, find the if-then-else block located just beyond the line commented as “C Initialize non-h2o tracers.” Read through this code and verify that it will initialize the constituent fields. Next, place comments in column on of the code within this if-block that would be executed if SETTRACE equals 'init', and replace it with code that is intended to perform the initialization.

2. Surface Fluxes, Sources, and Sinks

The user may wish to add surface fluxes and atmospheric sources and sinks that influence constituent values. Code to generate surface fluxes and atmospheric sources and sinks should be added to subroutine typhsac.F immediately prior to the call to vdintr.F. The computed surface flux tendencies (Kg/m²/sec) are to be stored in the array cf1x (local to tphysac.F) beginning at the array position cf1x(1, 2) for the first tracer, cf1x(1, 3) for the second tracer, etc. (the first column of this array, cf1x(1, 1), is reserved for the surface flux of water vapor). Sources and sinks are also to be computed as tendencies (Kg/Kgair/sec) and stored in the array srcsnk, beginning at the array position srcsnk(1, 1, 2) (with subsequent tracers stored at srcsnk(1, 1, 3),...
etc.). The user should not add these fluxes, sources, and sinks directly to the constituent fields themselves; the Model will do so.

Constituent surface flux tendencies can be written to a history file via namelist variables PRIMARY and/or AUXF (the field name “SFxx” which corresponds to the surface flux is already in the Master Field List). The source/sink tendency field, “SSxx,” is also in the Master Field List. However, to write it to a history file the user must insert a call to outf1d.F in the code (see “Adding New Variables” on page 154), and then use PRIMARY and/or AUXF.

3. Adjustment Processes

Code to perform adjustment physics (other than convective transport) to the time level n constituents should be inserted within routine aphys.F. Unlike the source/sink and flux calculations described above, there is no pre-existing code to do the adjustment. Users must supply this code (i.e. define arrays and common blocks, add to the Master field list, etc.).

4. Global Minima

During the evolution of a Model run, constituent values may fall below an “expected” minimum. The array qmin (PCNST), set in initcom.F and stored in common block /comqmrnin/, contains the minimum acceptable value for each constituent. The first element of this array, qmin (1), holds the minimum value for water vapor, and successive elements hold the minimum values for any new constituents. These minimum values in qmin are checked against Model-predicted values in routine qneg3.F, which is called at several instances throughout the code. The default value for water vapor is 1x10^{-12} (Kg/Kgair), and for each non-H2O constituent in qmin is 0.0 (Kg/Kgair).1 Currently, the Model will compare the constituent values with qmin at all points in the atmosphere. 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 constituents specified by the user. The qmin values may be reset by the user in the routine initcom.F. In cases where the Model is carrying a constituent that is not positive definite (e.g. potential vorticity), qmin may be set to a large negative value.

1. This is true except for one special case in routine vdiff.F, where qmincg (1) = 0.0 is used as the water vapor minimum instead of calling qneg3.F and using qmin (1).
D. Changing Resolution

The Model resolution can be changed from those provided with this distribution of the CCM3 (T5, T21, T31, and T42). In order to do so, the user is required to provide new initial and boundary datasets that match the new resolution, as well as set the appropriate parameters in params.h. Changes in horizontal resolution may also require changing physical parameterizations and/or changes in namelist variables, such as the timestep, diffusion coefficients, etc.

E. Adding Parameterizations

Before attempting to add your own parameterization to CCM3, it is necessary to read “Other Common Blocks and Header Files: The Parameterization Interface” on page 112, concerning the design of the parameterization interface. Adherence to the rules outlined in that section results in a parameterization that needs no information describing the overall Model data structures—all that is needed are longitude and level dimensions of input arguments on the Model grid. Latitude, time, and extended grid addressing are all handled by the Model control code.

Implementing parameterizations in this way requires that some restrictions be placed on the coding of the parameterizations. A parameterization should need no knowledge of the order in which Model fields are stored in memory. This implies that if a parameterization requires two time levels of a particular field, say temperature, these two levels are passed as two separate arrays by the interface routine. Parameterizations should also provide separate output arguments to return “answers” to the main Model; that is, they should not modify input arguments which point into the main Model buffer or other Model data structures. Time tendencies may be calculated from the input and output arguments at the level of the parameterization interface routine. Computed (or recomputed) fields should also be stored in the Model buffer by the interface routine.

The user also needs to be aware of restart requirements. If there are variables in your parameterization that need to be present at restart, they must be initialized at restart time. For new variables this means that they must appear on the regeneration dataset. If a new field is stored in the Model buffer, it will automatically be written to the regeneration dataset. Otherwise, you must add code to subroutine wrtrs1.F that will write your data to a regeneration file, as well as to subroutine splitf.F, so that the data can be read on a continuation run. In working with the restart logic in the Model, there are certain routines, most notably subroutines splitf.F and wrtrs1.F, in which blocks of code are essentially replicated for the in-core and out-of-core cases. Any changes you make in one block of this code must also be made in the corresponding block of code.
In adding a parameterization to the code, new subroutines and header files with common blocks are created. In order to incorporate these changes into the executable file, the Objects and Depends files used by the Makefile must be updated. Refer to “Modifying the Code” on page 151 for explicit directions about how to update these files.

F. Trouble-Shooting Model Changes

When a model run goes awry, where does one turn? First, if there is any doubt as to the reason for abnormal termination, ensure that the Model is run single-threaded. This is because abnormal termination in a multitasked job can result in confusing ancillary error messages that are generated as a result of the initial error with which the user is concerned.

If the Model terminates with an internally generated error message, the first place to look is “Model Error Messages” on page 74 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., “OK, fine, the dry adjustment procedure failed to converge, but why?”). 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 the way the user is changing certain prognostic variable calculations).

1. Resource Allocation Errors

If insufficient storage is allocated for scratch files located on the SSD, the Model will terminate and a somewhat informative error message will be printed if the file assign command was done with novfl specified. novfl instructs the operating system to abort if file I/O to the SSD exceeds the request as specified on the #QSUB -1Q line in the batch job script (see “Creating and Running the Executable in Batch Mode” on page 13). 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 -1Q request must at least match this number if novfl was specified. If novfl is not specified, scratch file I/O requests which exceed the #QSUB -1Q specification will spill to disk. Spilling to disk can outrageously degrade turnaround time for a long simulation.
An insufficient memory request (#QSUB -1M) may result in a reasonable error message of the form:

\[ \text{Not enough space} \]

or it may not. Check the line of segldr output which reads:

\[ \text{Program length- 22363070 octal 4843064 decimal} \]

You must ensure that the number of megawords requested on the #QSUB -1M 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. The reason is that during a run 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. The bottom line is: if the Model is using close to the amount of memory declared on the #QSUB -1M line and it appears to be dying due to some system error (e.g.: SAVDIS: Error in ishell call, ier= 256), try increasing the memory request by approximately 1 million words and rerun the job.

2. Coding Errors

One of the rules imposed by the CCM3 coding standard (see “CCM3 Coding Standard” on page 164) is the use of the Fortran implicit none statement in all program routines. This is done expressly to catch errors in variable usage, and is recommended for all user-written code for the same reason.

The flint utility is most useful for determining if there is an argument list mismatch between routines, when a variable is referenced before it is set, or if other serious coding errors such as common block length mismatches between routines exist. It also gives helpful hints about unused variables, variables that are set but never referenced, and the like.

Since flint looks at source code, the user can check his code for errors before even running it through the compiler. We recommend gathering all the source code into one file, then invoking flint with the -g and -x132 command line arguments. This utility can provide other information, such as a calling tree, or a symbol table with cross-reference list. The user is referred to the UNICOS man page for further details.

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 a user change to Model code 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 bounds checker

The -Rb command line argument to cft77 turns on array bounds checking. During model execution of code compiled with this option, a message will be printed whenever an array is referenced outside its declared dimensions. This option disables all vectorization, resulting in an enormous increase in CPU time (up to a factor of 100!) used for a given simulation. The -Rb option is therefore only practical if the Model is blowing up very early in the run, or if the user wants to turn it on in a very limited section of the code. In standard CCM3 code, arrays are never referenced outside of their declared bounds.

The symbolic debugger

The -ez option to the Cray cft77 compiler can also be helpful for run-time debugging of CCM3. This option causes debug symbol table information to be written to the object file produced by the compiler, which is then used by utilities to analyze the post-mortem core file after the Model crashes. Its usage results in a relatively small performance penalty (less than 10%) during execution, so it can feasibly be used when the Model doesn’t die until well into the simulation.

There are many utilities that use this symbol table information, (prof, cdbx, etc.), but the only one which will be discussed here is debug. This utility is most useful for reporting values of common variables and local variables at the time of Model termination. The -B option to debug requests that all common variables be printed. The -c1 option says to print local variables only one level deep into the branch of the calling tree which was active at the time of termination. The -d option can be used to determine how many values from each array dimension are actually printed when debug examines the user’s “core” file. Other options to debug are available, and again the user is referred to the man page for further details.

Presetting to “indefinite”

Statically allocated memory locations should always be initialized to “indefinite” during a run (the -f indef option to segldr). It costs virtually nothing to do this and will cause an error to be reported if uninitialized static memory is used in any floating-point computation. 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. In CCM3, 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%.
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 “Model Error Messages” on page 74). Where the Model died is not always the same place where things first started going wrong, however.

Some of the tools mentioned in the previous section (“Coding Errors” on page 162) can also be helpful if a formulation error is suspected. In particular, use of the -ez compiler option followed by the debug command is a very useful diagnostic tool if the Model is actually crashing.

Comparing history files

If the Model is running but producing incorrect or suspicious history files, a quick and easy-to-use diagnostic program, called cprtps, is available from the Core Group. This program provides a statistical analysis of differences in history file data. It requires two input history files with data valid for identical Model times. No user input is required. cprtps 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.

G. CCM3 Coding Standard

This section describes the coding standard that was established in creation of the CCM2. The CCM3 code adheres as much as possible to this standard. Any code that subsequently becomes a part of the Model should also follow the standard.

One reason for imposing a standard on CCM3 coding is to enhance the usefulness of certain system debugging tools. For instance, CCM3 may be run through the Fortran Lint (flint) global cross-reference and code checking tool on the Cray and produce intelligible output of a reasonable length. That is, there are few instances of nonstandard Fortran usage in the Model and, therefore, if you use this tool to debug your own Model changes, the output should be relevant to your code.

Another debugging tool to be used on the CCM is the Cray bounds checker. There are no constructs in the standard CCM3 code that generate large numbers of “warnings” or errors in using these tools. We strongly recommend that users adhere to this rule in their own code.

1. Code Appearance

All Fortran statements except comments and quoted strings will be coded using lower case letters. Comments should be in mixed case to help set them apart from the code. All character literals and strings, including those
in *format* statements, will be delimited by the single quote (apostrophe) character.

Executable code will be commented to explain the function of the statements following the comments. These comments (one or more lines) will be set off by blank comment lines ("C" in column 1 only). On those comment lines containing text, the text will start in column 3. It is suggested that comments denoting logical groupings of code, declarations, etc., will be offset with comment lines consisting of a "C" in column 1 and "-" characters in columns 2 through 72.

In cases where brief comments on the actual line of code clarifies the code, the "!" construct may be used, where there must be white space between the end of the statement and the "!.

Special forms of comments for declaration statements are described below.

Declarative statements

For each program routine, following the *subroutine* or *function* statement, a block of comments delineated by "C--..." comment lines will briefly describe the function of the routine. Following this, a block of comments labelled "Code history" indicates the originator of the current version of the routine and the Core Group member responsible for standardizing the code, along with relevant dates. This section may include any notes pertinent to revisions, usage, etc.

Each routine will contain a Fortran *implicit none* statement, i.e., all variables must be explicitly typed. Array dimensions will be specified via type statements, so there will be no *dimension* statements.

The main Model *parameter* statements and *common* statements will be stored as separate files to be "included" wherever appropriate in the code (see "Overview of Running CCM3" on page 3). Between each "include" directive should be a "C--..." comment line to provide a visual aid to separating *common* blocks, etc. The *common* variables and *parameters* should be vertically registered in fields of 8 characters, up to 5 on a line. Each block should start with a comment describing the functionality of the statement.

Subroutine arguments should be typed and described in a section titled "Arguments." Each argument declaration will be typed on a separate line, with a comment following a "!" character to briefly describe its function. Within this list, input arguments will appear first, following three comment lines, as shown in the following sample block of code. Input/output arguments will be treated similarly, followed by output arguments.
See “Code Structure” on page 167 for details about how argument lists should appear in subroutine and call statements. Any parameter statements that are local to a particular program routine will not be in an “include” file, but will appear in a section titled “Local Parameters”. The use of equivalence statements is discouraged, but if an equivalence should become necessary, it should be carefully commented. Fortran data statements are meant to assign an initial value to local variables only—those data statements setting common variables should appear in the block data procedure.

Following the “Code history” section, declarative statements should appear in the order implicit none, main Model parameter statements, common statements, type declarations of arguments, local parameters preceded by their type statements, type declarations of local variables, equivalence statements, data statements, save statements, external statements and statement functions.

The line continuation character used in column 6 throughout the code is the dollar sign ($).

2. Other Fortran Considerations

Printed output may be generated using format statements or as list-directed output (write (6,*)). For efficiency reasons, list-directed output which is repeated throughout the run should not be included on the printed output. However, it may be used to print diagnostic variables preceding an abnormal termination. Also, certain diagnostic printout may be included that is not intended to be a part of standard Model output. This printed output...
Changing the Model

should be under the control of a local logical variable rather than being commented-out in the code.

Fortran format statements containing scaling factors (e.g. 1p\texttt{e}12 . 3) should have a comma between the scaling character and the formatting character, as in “1p , e12 . 3”.

Fortran literals

Obscure “magic numbers” should not appear totally undocumented in Model calculations. These values should be assigned to local variables and commented as to their use.

3. Program Routine Names and Variable Names

There are no strict conventions for naming routines in the control portion of the Model. Where the basic function of a routine has not changed from that in CCM1, the name may remain the same, since users are familiar with the existing names. Routines within parameterizations, however, should all begin with the same two- or three-character designator, as in the CCM3 radiation parameterization (“rad”). The remaining 5 or 6 characters should be used as a mnemonic to indicate the general function of the routine.

Fortran parameter’s in the main Model parameter statements should begin with the letter “p”. This convention is also suggested for so-called “Local Parameters.” No specific naming conventions has been imposed for local variables.

4. Code Structure

Model code should use “structured programming” constructs where possible, avoiding the use of the go to statement and excessive use of statement labels. The do/end do construct may be used for short (up to 30 lines of code) do loops where it is easy to see the extent of the loop without paging through the code. Both do loops and if-then-else blocks should be consecutively indented at least two spaces at a time. Where part of a calculation appears on a continuation statement, the code should be indented past the equal sign if possible. A continuation line should never begin in a column preceding where the initial line of the statement begins. A single blank should appear on either side of the equal sign, and on either side of all + and - signs not enclosed in parentheses.

Statement numbers should monotonically increase within a program routine, by an increment of at least 10. Labels of 9000 and above are reserved for format statements. Statement labels should be right-justified to column 5. Also, the format statements should appear at the end of a program routine, following the return statement.

The argument lists in Fortran subroutine and call statements will be formatted as follows. Multiple arguments in a list should appear in the order
Changing the Model

input, input/output and output arguments. Arguments should be vertically registered, left-justified in an eight-character field. Argument lists for a particular subroutine and all call statements referencing it should line up and register correspondingly, if possible. If not, each additional line required should be indented consecutively, as shown in the following block of code, so that the corresponding argument can be found without counting all arguments up to that point.

```
subroutine x(a,b,c,d,e,f,g)

call x(pdela(index1,index2), pdelb(index1,index2),
carray(index3,index4), dummy(index5,index6),
ework(index7),
f(il,i2),g(il,i2))
```

Arrays passed through the argument list should be dimensioned according to the actual number of elements referenced within the calling routine, i.e., do not dimension dummy arrays by 1. If the dimension is not known (say, in a general-purpose utility routine) it may be specified by a *. This allows the code to run using the system bounds checker without generating many bogus errors.
References


Appendix A: Glossary

branch run— Refers to a type of continuation run of the CCM. A branch run starts a new case using the regeneration files from a previous run of the Model in place of its initial dataset.

case — A term used to denote a CCM3 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.

CCM1 — A previous version of the NCAR Community Climate Model, the basic code from which CCM2 was built.

CCM2 — The version of the NCAR Community Climate Model from which the CCM3 is descendent.

CCM3 — The current version of the CCM, which is documented in this edition of the User's Guide.

CCM Processor — A program available from the CCM Core Group which post-processes CCM history files on the NCAR computing system.

CSM — Climate System Model.

cf77 — Cray Fortran 77 compiler.

cpp — C preprocessor.

CPU — Central processing unit.

continuation run — Refers to a type of run that uses a regeneration file from a previous run to initialize the data fields (no initial dataset is read). Restart, regeneration, and branch runs are each possible types of a continuation run.

ECMWF — European Center for Medium Range Weather Forecasts

Fortran-callable — Describes a UNICOS shell command for which there exists a Fortran interface, i.e., a system function or subroutine callable from Fortran.
FFT— Fast Fourier Transform.

fmp — Fortran mid-processor.

frozen — Describes the Community Climate Model in its standard form, as documented in this User’s Guide and presented to the community.

header file — A file containing common blocks and variable declarations, named with a “.h” suffix (e.g. comhst.h). A header file may be included in any routine that need to access the variables it contains.

heap — Memory that is dynamically allocated by the system. Unlike stack memory, heap memory can be allocated or deallocated at any point during program execution.

history file — A binary dataset that is the primary output for field values generated by CCM3 during the course of a time integration. A CCM3 history file is a disk file with formatting identical to a CCM2 history tape.

history tape — A history file that has been disposed to the NCAR Mass Storage System.

in-core — Refers to the use of main memory during a Model run.

initialization run— Refers to an initial run of the CCM. A initialization run starts a new case using the specified initial and boundary condition datasets.

LSM— Land Surface Model.

Makefile — A file used by the UNIX make utility that directs preprocessing, compiling, and loading of CCM3 sources files into an executable file.

Megabyte (Mb) — $2^{20}$ bytes (1,048,576 bytes; 1 million bytes ≈ 0.95 Mb).

Megaword (Mw) — 1,048,576 Cray words (1 word = 8 bytes = 64 bits).

MPP— Massively Parallel Processors.

MSS — The NCAR Mass Store System, consisting of the IBM 9672 Mass Storage Control Processor (MSCP), a large IBM disk farm for intermediate storage, an IBM 3490 Cartridge Tape System for archival storage and a Storage Tek Automatic Cartridge System (robot).
**multitasked** — Refers to a program configured to execute on more than one central processor simultaneously.

**namelist** — A Fortran 77 extension that processes input parameters in a free-form fashion.

**out-of-core** — Refers to the use of secondary storage for cycling to and from main memory during a Model run.

**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, 3 or 4 to 1, via system routine `packed`.

**pcnst** — Number of constituents carried in the model.

**plug-compatible** — Refers to a parameterization coding standard to promote ease of replacement and/or exchange of parameterizations in the Model.

**plat** — Number of Gaussian latitudes on the transform grid.

**plev** — Refers to the number of vertical model levels.

**plon** — Number of longitudes on the transform grid.

**pointer** — An integer variable, defined by a Fortran `pointer` statement, that references an address in main memory.

**preprocessor** — A system program which takes input directives and source code and builds a file ready for a language compiler.

**PVM** — Parallel Virtual Machine.

**restart run** — Refers to 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 and regeneration datasets.

**regeneration run** — Refers to a type of continuation run of the CCM. A regeneration run is typically used to repeat part of a previous case in order to restore corrupt history files. A regeneration run is started by using data from specified regeneration files of the case being regenerated.

**single-threaded** — Refers to code that executes on only one processor at a time.

**SDS** — Secondary Data Segment, a file management method available via UNICOS software for use on the SSD.

**SLT** — Semi-Lagrangian Transport.
SOM — Slab Ocean Model.

SSD — Solid-State Storage Device, a very high-performance secondary storage device on the Cray Y-MP.

stack — An area of memory where storage for variables is allocated when a subprogram or procedure begins execution and is released when execution completes.

tar file — A single file that archives many related CCM3 files and directories using the UNIX `tar` utility.

time sample — Refers to the history fields output on a given timestep. One disk history file will contain MFILT(i) time samples.

transform grid — Refers to a finite difference-like grid which is used to evaluate all nonlinear and diabatic forcing terms in physical space.

timestep — Refers to a single time integration of the Model from one value of nstep to the next.

word — 4-byte (32-bit) or 8-byte (64-bit) sequence; machine dependent. One Cray word is 64 bits.
Appendix B: CCM3 Calling Tree

The figures in this section depict the CCM3 calling tree, showing all program modules except single-purpose utility routines. These charts read in order of subroutine calls from left to right. The list below shows a short index relating the figure number to a brief description of the figure.

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. B.1.a</td>
<td>Initializations, part (a)</td>
</tr>
<tr>
<td>Fig. B.1.b</td>
<td>Initializations, part (b)</td>
</tr>
<tr>
<td>Fig. B.2</td>
<td>Time-stepping loop</td>
</tr>
<tr>
<td>Fig. B.3.a</td>
<td>Pre-coupler Gaussian latitude scan, part (a)</td>
</tr>
<tr>
<td>Fig. B.3.b</td>
<td>Pre-coupler Gaussian latitude scan, part (b)</td>
</tr>
<tr>
<td>Fig. B.4</td>
<td>Ocean and sea ice models</td>
</tr>
<tr>
<td>Fig. B.5</td>
<td>Land surface model</td>
</tr>
<tr>
<td>Fig. B.6</td>
<td>Post-coupler Gaussian latitude scan</td>
</tr>
<tr>
<td>Fig. B.7</td>
<td>Semi-Lagrangian transport</td>
</tr>
<tr>
<td>Fig. B.8</td>
<td>Spectral routines and conversion to grid-point space</td>
</tr>
</tbody>
</table>

Note the main time-stepping loop is shown in Figure B.2. In this figure, all of the CCM3 routines coded within a multitasked loop are shown in the shaded boxes.
Figure B.1.a. Calling tree: Initializations, part (a)
Figure B.1.b. Calling tree: Initializations, part (b)
Figure B.2. Calling tree: Time-stepping loop
Figure B.3.a. Calling tree: Pre-coupler Gaussian latitude scan, part (a)
Figure B.3.b. Calling tree: Pre-coupler Gaussian latitude scan, part (b)
Figure B.4. Calling tree: Ocean and sea ice models
Figure B.5. Calling tree: Land surface model
Figure B.6. Calling tree: Post-coupler Gaussian latitude scan
Figure B.7. Calling tree: Semi-Lagrangian transport
Figure B.8. Calling tree: Spectral routines and conversion to grid-point space
This appendix contains the complete printed output from a 72 timestep CCM3 run. This output was generated from a batch run of the standard CCM3 with spectral dynamics, at T42 resolution, on an NCAR Cray. The namelist input file used is shown in “Example 1 — Initial Run, One Day” on page 29. Selected portions of the printout are explained in “Model Printed Output” on page 69.
NCAR Community Climate Model (CCM3), Version 1.0 [05/15/96], Copyright (C) 1996
University Corporation for Atmospheric Research
All Rights Reserved

(Note: Online documentation is available on the CCM3 home page: http://www.cgd.ucar.edu/cms/ccm3/)

DATE 05/06/96 TIME 16:03:57

E&CCMEXP
CASEID = 'clsst01'
CTITLE = 'Test run: climatological SST'
NCDATA = 'SEP1.T42.0596'
BNDTI = 'tibds.T42.0596'
BNDTVS = 'STR-sst.T42.0596'
BNDTVO = 'ozn.T42.0596'
NSWRPS = 'passwd'
NSVSN = 'rstrt'
IRT = 100
NSREST = 0
NDENS = 2
NNBDAT = 000901
NNBSEC = 0
NNDIAS = 0
NNSBAS = 0
MFILT = 20
DTIME = 1200.
NESTEP = 72
NHTFRQ = 72
INCORBUF = .false.
INCORHST = .false.
INCORRAD = .false.

&END

DATA: Prepending /ccm to mss paths for local disk path to search for datasets

*** INPUT VARIABLES (CCMEXP) ***

Initial run
********** CASE = clsst01 **********
Test run: climatological SST
User name to build pathnames = jquser
Initial dataset is: SEP1.T42.0596
Time-invariant boundary dataset is: tibds.T42.0596
Time-variant boundary dataset (sst) is: STR-sst.T42.0596
Time-variant boundary dataset (ozone) is: ozn.T42.0596
Restart dataset is: rstrt
Maximum size of regeneration files is: 380000000 bytes
Write passwd for output tapes (NSWRPS) is passwd
Restart flag (NSREST) 0=no,1=yes,2=regen 0
Retention time for History Tapes = 100 days
Virtual MS volume for History Tape (NSMVN) CTPUBLIC
Retention time for restart and regeneration tapes = 100 Days
Virtual MS volume for Regen data (NRMVN) CTPUBLIC
History File 1 packing density = 2
History File 1 write frequency (NHTFRQ) 72
Number of time samples per history file (MFILT) 20
Fields on history file 1 are averaged.
Regeneration data will be written for every 1 history files
Base day,seconds of day = 0 0
Base date,seconds of date = 901 0
Time step to end run (NESTEP) 72
Time step in seconds (DTIME) 1200.
Time filter coefficient (EPS) 0.060
DEL2 Horizontal diffusion coefficient (DIF2) 0.250E+06
DEL4 Horizontal diffusion coefficient (DIF4) 0.100E+17
Number of levels Courant limiter applied 1
Lowest level for dry adiabatic adjust (NLVDRY) 3
Frequency of Shortwave Radiation calc. (IRADSW) 3
Frequency of Longwave Radiation calc. (IRADLW) 3
Frequency of Absorptivity/Emissivity calc. (IRADAE) 36
Frequency of SST Initialization calc. (ITSSST) 1
SST dataset will be reused for each model year
OZONE dataset will be reused for each model year
Output files will be disposed ASYNCHRONOUSLY
Diurnal averaging will not be done
divergence damper NOT invoked

-----------------------------
Opened scratch direct access unit nral= 11 size=491520 words.
Opened scratch direct access unit nrbl= 21 size=1736704 words.
Opened scratch direct access unit nsplit=70 size=917504 words.
Opened scratch direct access unit nabem= 60 size=3801088 words.
-----------------------------
TRUNC: n cutoff for GRCALC vectorization = 36
ATCHBND:Successfully opened file SEP1.T42.0596 on unit 4
Machine epsilon = 7.11E-15

<table>
<thead>
<tr>
<th>Layer</th>
<th>Locations (*1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.9170</td>
</tr>
<tr>
<td>2</td>
<td>7.9292</td>
</tr>
<tr>
<td>3</td>
<td>21.5539</td>
</tr>
<tr>
<td>4</td>
<td>49.1834</td>
</tr>
<tr>
<td>5</td>
<td>83.1425</td>
</tr>
<tr>
<td>6</td>
<td>79.9308</td>
</tr>
<tr>
<td>7</td>
<td>75.7738</td>
</tr>
<tr>
<td>8</td>
<td>70.5752</td>
</tr>
<tr>
<td>9</td>
<td>64.2963</td>
</tr>
<tr>
<td>10</td>
<td>56.9838</td>
</tr>
<tr>
<td>11</td>
<td>48.7913</td>
</tr>
<tr>
<td>12</td>
<td>39.9895</td>
</tr>
<tr>
<td>13</td>
<td>30.9631</td>
</tr>
<tr>
<td>14</td>
<td>22.1902</td>
</tr>
<tr>
<td>15</td>
<td>14.2039</td>
</tr>
<tr>
<td>16</td>
<td>7.5413</td>
</tr>
<tr>
<td>17</td>
<td>2.6838</td>
</tr>
<tr>
<td>18</td>
<td>0.0000</td>
</tr>
<tr>
<td>19</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Reference pressures (Pa)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Reference Pressures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>291.700</td>
</tr>
<tr>
<td>2</td>
<td>792.9200</td>
</tr>
<tr>
<td>3</td>
<td>1000.0000</td>
</tr>
<tr>
<td>4</td>
<td>1000.0000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>2155.3900</td>
</tr>
<tr>
<td>4</td>
<td>4918.3400</td>
</tr>
<tr>
<td>5</td>
<td>8314.2500</td>
</tr>
<tr>
<td>6</td>
<td>11798.4900</td>
</tr>
<tr>
<td>7</td>
<td>16308.2600</td>
</tr>
<tr>
<td>8</td>
<td>21947.8900</td>
</tr>
<tr>
<td>9</td>
<td>28759.5900</td>
</tr>
<tr>
<td>10</td>
<td>36692.4400</td>
</tr>
<tr>
<td>11</td>
<td>45580.0900</td>
</tr>
<tr>
<td>12</td>
<td>55128.7200</td>
</tr>
<tr>
<td>13</td>
<td>64920.9600</td>
</tr>
<tr>
<td>14</td>
<td>74438.2900</td>
</tr>
<tr>
<td>15</td>
<td>83102.1200</td>
</tr>
<tr>
<td>16</td>
<td>90330.0300</td>
</tr>
<tr>
<td>17</td>
<td>95599.7500</td>
</tr>
<tr>
<td>18</td>
<td>98511.2200</td>
</tr>
<tr>
<td>19</td>
<td>100000.0000</td>
</tr>
</tbody>
</table>

Truncation Parameters
NTRM = 42
NTRN = 42
NTRK = 42

REFERENCE TEMPERATURES FOR SEMI-IMPLICIT SCHEME =
300.000 300.000 300.000 300.000 300.000 300.000 300.000 300.000 300.000 300.000 300.000

GRAVITY WAVE PHASE SPEEDS (M/S) FOR MEAN STATE =
8.140 6.062 4.359 2.961 1.805 0.839

GRAVITY WAVE EQUIVALENT DEPTHS (M) FOR MEAN STATE =
11983.449 4416.865 1648.486 762.414 394.846 225.014 136.466 84.618 -52.389 32.142 19.514 11.649
6.758 3.747 1.938 0.894 0.332 0.072

INIDAT: MASS OF INITIAL DATA BEFORE CORRECTION = 1.0040496620E+04
DRY MASS WILL BE HELD = 1.0016357065E+04
MASS OF MOISTURE AFTER REMOVAL OF NEGATIVES = 2.4058690524E+01

-----------------------------
RAYLIEGH FRICTION

BOTTOM LEVEL: NBOTRL= 0 NTOPRL= 1

COEFFICIENTS
0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00

*** SATURATION VAPOR PRESSURE TABLE COMPLETED ***

ATCHBND: Successfully opened file tibds.T42.0596 on unit 1
ATCHBND: Successfully opened file STR-sst.T42.0596 on unit 3
ATCHBND: Successfully opened file ozn.T42.0596 on unit 2

OZNINI: Read ozone data for date (ymmddd) 840116
OZNINI: Read ozone data for date (ymmddd) 840214
OZNINI: Read ozone data for date (ymmddd) 840316
OZNINI: Read ozone data for date (ymmddd) 840415
OZNINI: Read ozone data for date (ymmddd) 840516
OZNINI: Read ozone data for date (ymmddd) 840615
OZNINI: Read ozone data for date (ymmddd) 840716
OZNINI: Read ozone data for date (ymmddd) 840816
OZNINI: Read ozone data for date (ymmddd) 840915

***********************************************************************************************

LSM land surface model, version 1

initial run at:
    nstep= 0 year= 0 month= 9 day= 1 seconds= 0

time step = 1200. s except for soil hydrology which uses a 600. s time step

multi-processing: there will be 39 calls to the vector-based code
    1 calls with vector length 157
    38 calls with vector length 154

preset fortran unit numbers:
unit 5 = standard input
unit 6 = standard output

logical run control parameters:
use initial data set = F
prognostic hydrology = T
flag for random perturbation test = F
energy and water conservation checks = F

data files:
surface data = surfdat.T42.0596
initial data = arbitrary initialization

gird characteristics:
latitude points = 64
maximum longitude points = 128
minimum longitude points = 128
land points on grid = 2796
total land points including subgrid points = 6009

successful set up of lsmtc common block
successful set up of lsmtv common block
successful set up of LSM

*****************************
SSTINI: Read sst data for date (yymmdd) 116
SSTINI: Read sst data for date (yymmdd) 214
SSTINI: Read sst data for date (yymmdd) 316
SSTINI: Read sst data for date (yymmdd) 415
SSTINI: Read sst data for date (yymmdd) 516
SSTINI: Read sst data for date (yymmdd) 615
SSTINI: Read sst data for date (yymmdd) 716
SSTINI: Read sst data for date (yymmdd) 816
SSTINI: Read sst data for date (yymmdd) 915
Opened scratch direct access unit sunit= 22 size=4194304 words

Size of out-of-core files:
Model buffer = 2228224
Exbuf = 917504
History buffer = 4194304
Abs/ems = 3801088
--------
Total = 11141120
Size of files held in core:

        --------
Total      = 0

**** Summary of Logical Unit assignments ****

Initial dataset unit (ninit) = 4
Time-inv boundary dataset (nbndti) = 1
Ozone dataset unit (nozone) = 2
SST dataset unit (nsst) = 3
Model buffer scratch file (nrla) = 11
Model buffer scratch file (nrbl) = 21
Exbuf scratch file (nsplit) = 22
History buffer scratch file (sunit) = 60
Abs/ems scratch file (nabem) = 70
History file number 1 = 20
Restart dataset unit (nsds) = 7
Master regeneration unit (nrg) = 8
Regeneration dataset units (nrgl) = 9 10 12 13
Abs/ems unit for restart (nrg2) = 15 16 17 18
Regeneration units for hist file 1 = 23 24 25 26 27

NSTEP RMSZ
NSTEP = 0 8.85216994233591E-05

sending string to ishell from prealc:
setf -n 351735984 h0001
SCAN1: Successfully opened file h0001
hunit(ktape) = 20

*** HEADER FOR CCM3 HISTORY FILE ***

*** Primary History File ***

CASE: clsst01
TITLE: Test run: climatological SST

LENHDI MFTYP MFILH MFILTH NRBD MAXSIZ NDAVU NXXX NLO N LON N LONW
292 43 1 20 3 33662 65282 0 128 128

PLAT PLEV PTRM PTRN PTRK NFLDH NSTEPH NSTPRH NITSLF NDBASE
## Field List

<table>
<thead>
<tr>
<th>FLD NO.</th>
<th>NAME</th>
<th>FLG.</th>
<th>FLD PT.</th>
<th>PACK.</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q</td>
<td>12</td>
<td>3</td>
<td>2</td>
<td>KG/KG</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>12</td>
<td>1191</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>3</td>
<td>U</td>
<td>12</td>
<td>2379</td>
<td>2</td>
<td>M/S</td>
</tr>
<tr>
<td>4</td>
<td>V</td>
<td>12</td>
<td>3567</td>
<td>2</td>
<td>M/S</td>
</tr>
<tr>
<td>5</td>
<td>PS</td>
<td>10</td>
<td>4755</td>
<td>2</td>
<td>PA</td>
</tr>
<tr>
<td>6</td>
<td>TS</td>
<td>10</td>
<td>4821</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>7</td>
<td>DC01</td>
<td>12</td>
<td>4887</td>
<td>2</td>
<td>KG/KGS</td>
</tr>
<tr>
<td>8</td>
<td>TA01</td>
<td>12</td>
<td>6075</td>
<td>2</td>
<td>KG/KGS</td>
</tr>
<tr>
<td>9</td>
<td>TS1</td>
<td>10</td>
<td>7263</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>10</td>
<td>TS2</td>
<td>10</td>
<td>7329</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>11</td>
<td>TS3</td>
<td>10</td>
<td>7395</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>DMI</td>
<td>10</td>
<td>7461</td>
<td>2</td>
<td>UG/M2S</td>
</tr>
<tr>
<td>13</td>
<td>VD01</td>
<td>12</td>
<td>7527</td>
<td>2</td>
<td>KG/KGS</td>
</tr>
<tr>
<td>14</td>
<td>TS4</td>
<td>10</td>
<td>8715</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>15</td>
<td>CGS</td>
<td>11</td>
<td>8781</td>
<td>2</td>
<td>S/M2</td>
</tr>
<tr>
<td>16</td>
<td>TAH</td>
<td>10</td>
<td>9969</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>17</td>
<td>DTH</td>
<td>12</td>
<td>10035</td>
<td>2</td>
<td>K/S</td>
</tr>
<tr>
<td>18</td>
<td>FRM</td>
<td>10</td>
<td>11223</td>
<td>2</td>
<td>UMOL/M2S</td>
</tr>
<tr>
<td>19</td>
<td>KVKH</td>
<td>11</td>
<td>11289</td>
<td>2</td>
<td>M2/S</td>
</tr>
<tr>
<td>20</td>
<td>FCO2</td>
<td>10</td>
<td>12477</td>
<td>2</td>
<td>UMOL/M2S</td>
</tr>
<tr>
<td>21</td>
<td>DTV</td>
<td>12</td>
<td>12543</td>
<td>2</td>
<td>K/S</td>
</tr>
<tr>
<td>22</td>
<td>QRL</td>
<td>12</td>
<td>13731</td>
<td>2</td>
<td>K/S</td>
</tr>
<tr>
<td>23</td>
<td>ORO</td>
<td>0</td>
<td>14919</td>
<td>2</td>
<td>FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>EFFCLD</td>
<td>12</td>
<td>14985</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>QRS</td>
<td>12</td>
<td>16173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>CMFDT</td>
<td>12</td>
<td>17361</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>CMFDQ</td>
<td>12</td>
<td>18549</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>CLDHGH</td>
<td>10</td>
<td>19737</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>RSW</td>
<td>10</td>
<td>19803</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>ETADOT</td>
<td>11</td>
<td>19869</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>BEVAP</td>
<td>10</td>
<td>21057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>CLDLOW</td>
<td>10</td>
<td>21123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>FMICR</td>
<td>10</td>
<td>21189</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>CLDMED</td>
<td>10</td>
<td>21255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>CMFMC</td>
<td>11</td>
<td>21321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>FCEV</td>
<td>10</td>
<td>22509</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>LHFLX</td>
<td>10</td>
<td>22575</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>PBLH</td>
<td>10</td>
<td>22641</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>CLDTOT</td>
<td>10</td>
<td>22707</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>OMEGA</td>
<td>12</td>
<td>22773</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>FGEV</td>
<td>10</td>
<td>23961</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>QDRAI</td>
<td>10</td>
<td>24027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>BTTRAN</td>
<td>10</td>
<td>24093</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>PRECL</td>
<td>10</td>
<td>24159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>PRECC</td>
<td>10</td>
<td>24225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>DTCOND</td>
<td>12</td>
<td>24291</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>CNVCLD</td>
<td>10</td>
<td>25479</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>FRMF</td>
<td>10</td>
<td>25545</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>SHFLX</td>
<td>10</td>
<td>25611</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>QINFL</td>
<td>10</td>
<td>25677</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>FCTR</td>
<td>10</td>
<td>25743</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>FSDS</td>
<td>10</td>
<td>25809</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>TREFH</td>
<td>10</td>
<td>25875</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>TAUGNX</td>
<td>10</td>
<td>25941</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>TAUGNW</td>
<td>10</td>
<td>26007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>TGRD</td>
<td>10</td>
<td>26073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>NDVI</td>
<td>10</td>
<td>26139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>FLNS</td>
<td>10</td>
<td>26205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>CLOUD</td>
<td>12</td>
<td>26271</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>FLNSC</td>
<td>10</td>
<td>27459</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>PHIS</td>
<td>0</td>
<td>27525</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>FLNT</td>
<td>10</td>
<td>27591</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>FLNTC</td>
<td>10</td>
<td>27657</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>TVEG</td>
<td>10</td>
<td>27723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>SOLIN</td>
<td>10</td>
<td>27789</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>FPSN</td>
<td>10</td>
<td>27855</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>QPERT</td>
<td>10</td>
<td>27921</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>68</td>
<td>FSNS</td>
<td>10</td>
<td>27987</td>
<td>2</td>
<td>W/M²</td>
</tr>
<tr>
<td>69</td>
<td>FSNSC</td>
<td>10</td>
<td>28053</td>
<td>2</td>
<td>W/M²</td>
</tr>
<tr>
<td>70</td>
<td>QFLX</td>
<td>10</td>
<td>28119</td>
<td>2</td>
<td>KG/M²/S</td>
</tr>
<tr>
<td>71</td>
<td>TPERT</td>
<td>10</td>
<td>28185</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>72</td>
<td>FSNT</td>
<td>10</td>
<td>28251</td>
<td>2</td>
<td>W/M²</td>
</tr>
<tr>
<td>73</td>
<td>FSNTC</td>
<td>10</td>
<td>28317</td>
<td>2</td>
<td>W/M²</td>
</tr>
<tr>
<td>74</td>
<td>QOVER</td>
<td>10</td>
<td>28383</td>
<td>2</td>
<td>MM/S</td>
</tr>
<tr>
<td>75</td>
<td>ZMDQ</td>
<td>12</td>
<td>28449</td>
<td>2</td>
<td>KG/KGS</td>
</tr>
<tr>
<td>76</td>
<td>USTAR</td>
<td>10</td>
<td>29637</td>
<td>2</td>
<td>M/S</td>
</tr>
<tr>
<td>77</td>
<td>SRFRAD</td>
<td>10</td>
<td>29703</td>
<td>2</td>
<td>W/M²</td>
</tr>
<tr>
<td>78</td>
<td>ZMDT</td>
<td>12</td>
<td>29769</td>
<td>2</td>
<td>K/S</td>
</tr>
<tr>
<td>79</td>
<td>TSOI3</td>
<td>10</td>
<td>30957</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>80</td>
<td>TSOI4</td>
<td>10</td>
<td>31023</td>
<td>2</td>
<td>K</td>
</tr>
<tr>
<td>81</td>
<td>TAUX</td>
<td>10</td>
<td>31089</td>
<td>2</td>
<td>N/M²</td>
</tr>
<tr>
<td>82</td>
<td>TAUY</td>
<td>10</td>
<td>31155</td>
<td>2</td>
<td>N/M²</td>
</tr>
<tr>
<td>83</td>
<td>SNOWH</td>
<td>12</td>
<td>31221</td>
<td>2</td>
<td>M</td>
</tr>
<tr>
<td>84</td>
<td>UTGW</td>
<td>12</td>
<td>31287</td>
<td>2</td>
<td>M/S²</td>
</tr>
<tr>
<td>85</td>
<td>VTGW</td>
<td>12</td>
<td>32475</td>
<td>2</td>
<td>M/S²</td>
</tr>
</tbody>
</table>

**SIGMA VALUES AT FULL LEVELS**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.809300000000002E-03</td>
<td>1.30731000000001E-02</td>
<td>3.25591000000001E-02</td>
<td>6.39471000000000E-02</td>
<td>9.04320000000001E-02</td>
<td></td>
</tr>
<tr>
<td>1.38712900000000E-01</td>
<td>1.89190800000000E-01</td>
<td>5.98248200000000E-01</td>
<td>9.70445699999999E-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.01275499999998E-01</td>
<td>5.98248200000000E-01</td>
<td>9.70445699999999E-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.29275499999999E-01</td>
<td>9.70445699999999E-01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.29275499999999E-01</td>
<td>9.70445699999999E-01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SIGMA VALUES AT HALF LEVELS**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.91700000000000E-03</td>
<td>7.929200000000003E-03</td>
<td>2.15539000000000E-02</td>
<td>4.91834000000000E-02</td>
<td>8.31425000000001E-02</td>
<td></td>
</tr>
<tr>
<td>1.17984900000000E-01</td>
<td>1.63082600000000E-01</td>
<td>2.19478900000000E-01</td>
<td>2.87595899999999E-01</td>
<td>3.66924400000000E-01</td>
<td></td>
</tr>
<tr>
<td>4.55800900000000E-01</td>
<td>5.98248200000000E-01</td>
<td>6.49209599999999E-01</td>
<td>7.44382899999999E-01</td>
<td>8.31021199999999E-01</td>
<td></td>
</tr>
<tr>
<td>9.03300299999999E-01</td>
<td>9.58974999999999E-01</td>
<td>9.85212000000000E-01</td>
<td>1.00000000000000E+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LATITUDES**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23.72017</td>
<td>26.51077</td>
<td>29.30136</td>
<td>32.09194</td>
<td>34.88252</td>
<td>37.67309</td>
</tr>
<tr>
<td>51.62573</td>
<td>54.41620</td>
<td>57.20663</td>
<td>59.99702</td>
<td>62.78735</td>
<td>65.57761</td>
</tr>
</tbody>
</table>
GAUSSIAN WEIGHTS

<table>
<thead>
<tr>
<th>NSTEP</th>
<th>1.78328072146580E-03</th>
<th>4.14703326043206E-03</th>
<th>6.5044579686448E-03</th>
<th>8.84675982637578E-03</th>
<th>1.11681394601145E-02</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.36430478967601E-02</td>
<td>1.57260304760191E-02</td>
<td>2.7951715756741E-02</td>
<td>2.0134821534985E-02</td>
<td>2.22701738083598E-02</td>
</tr>
<tr>
<td></td>
<td>2.43527025686925E-02</td>
<td>2.63774679150316E-02</td>
<td>2.83396726142516E-02</td>
<td>3.02346570273731E-02</td>
<td>3.2079283548366E-02</td>
</tr>
<tr>
<td></td>
<td>3.38051618371289E-02</td>
<td>3.5472123526863E-02</td>
<td>3.70551258402401E-02</td>
<td>3.8550153178603E-02</td>
<td>3.99537411327251E-02</td>
</tr>
<tr>
<td></td>
<td>4.1262563246233E-02</td>
<td>4.24735151236550E-02</td>
<td>4.35873245293726E-02</td>
<td>4.45905856137576E-02</td>
<td>4.54916279274185E-02</td>
</tr>
<tr>
<td></td>
<td>4.6284765813196E-02</td>
<td>4.69681828162118E-02</td>
<td>4.75401657148335E-02</td>
<td>4.79993859564654E-02</td>
<td>4.83447622348090E-02</td>
</tr>
<tr>
<td></td>
<td>8.8574674415084E-02</td>
<td>8.8699597091461E-02</td>
<td>8.8574674415084E-02</td>
<td>8.8574674415084E-02</td>
<td>8.8574674415084E-02</td>
</tr>
<tr>
<td></td>
<td>7.9993858964645E-02</td>
<td>7.9401651748335E-02</td>
<td>6.9681828162118E-02</td>
<td>6.2847965813196E-02</td>
<td>5.44916279274185E-02</td>
</tr>
<tr>
<td></td>
<td>4.59058581637576E-02</td>
<td>4.35873245293726E-02</td>
<td>4.24735151236550E-02</td>
<td>4.1262563246233E-02</td>
<td>3.99537411327251E-02</td>
</tr>
<tr>
<td></td>
<td>3.8505131786032E-02</td>
<td>3.70551258402401E-02</td>
<td>3.5472123526863E-02</td>
<td>3.80516183712899E-02</td>
<td>3.2079283548366E-02</td>
</tr>
<tr>
<td></td>
<td>3.02346570273731E-02</td>
<td>2.83396726142516E-02</td>
<td>2.63774679150316E-02</td>
<td>2.43527025686925E-02</td>
<td>2.22701738083598E-02</td>
</tr>
<tr>
<td></td>
<td>2.0134821534985E-02</td>
<td>1.79517157756741E-02</td>
<td>1.57260304760191E-02</td>
<td>1.34630478967601E-02</td>
<td>1.11681394601145E-02</td>
</tr>
</tbody>
</table>

REFERENCE TEMPERATURES FOR SEMI-IMPPLICIT SCHEME =

<table>
<thead>
<tr>
<th></th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
<th>300.000</th>
</tr>
</thead>
</table>

GRAVITY WAVE PHASE SPEEDS (M/S) FOR MEAN STATE =

<table>
<thead>
<tr>
<th></th>
<th>342.800</th>
<th>208.117</th>
<th>127.143</th>
<th>86.466</th>
<th>46.974</th>
<th>36.582</th>
<th>28.806</th>
<th>22.666</th>
<th>17.753</th>
<th>13.833</th>
</tr>
</thead>
</table>

GRAVITY WAVE EQUIVALENT DEPTHS (M) FOR MEAN STATE =

<table>
<thead>
<tr>
<th></th>
<th>11983.449</th>
<th>4416.865</th>
<th>1648.486</th>
<th>762.414</th>
<th>394.846</th>
<th>225.014</th>
<th>136.466</th>
<th>84.618</th>
<th>52.389</th>
<th>32.142</th>
<th>19.514</th>
<th>11.649</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSTEP</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>9.84595E+04</td>
<td>2.413353235262298E+01</td>
<td>0.92</td>
<td>0.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>9.84596E+04</td>
<td>2.414295031287395E+01</td>
<td>0.91</td>
<td>0.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>9.84596E+04</td>
<td>2.415135721909280E+01</td>
<td>0.91</td>
<td>0.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>9.84579E+04</td>
<td>2.416027239636719E+01</td>
<td>0.91</td>
<td>0.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>9.84600E+04</td>
<td>2.417085286157116E+01</td>
<td>0.91</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>9.84601E+04</td>
<td>2.417963452281708E+01</td>
<td>0.91</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>9.84600E+04</td>
<td>2.418939734935675E+01</td>
<td>0.90</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>9.84601E+04</td>
<td>2.419931043290010E+01</td>
<td>0.90</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>9.84602E+04</td>
<td>2.420894125157531E+01</td>
<td>0.90</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>9.84603E+04</td>
<td>2.421876025314808E+01</td>
<td>0.90</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>9.84604E+04</td>
<td>2.422831696380877E+01</td>
<td>0.90</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>9.84605E+04</td>
<td>2.423759767966294E+00</td>
<td>0.90</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>9.84606E+04</td>
<td>2.424763134257262E+00</td>
<td>0.90</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>9.84607E+04</td>
<td>2.425603513169062E+02</td>
<td>0.89</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>9.84608E+04</td>
<td>2.426569239487887E+01</td>
<td>0.89</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>9.84609E+04</td>
<td>2.427411242872707E+01</td>
<td>0.89</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>9.84610E+04</td>
<td>2.428215406751247E+00</td>
<td>0.89</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>9.84611E+04</td>
<td>2.429081988821031E+01</td>
<td>0.89</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>9.84611E+04</td>
<td>2.429905145800285E+00</td>
<td>0.89</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>9.84612E+04</td>
<td>2.430582597302362E+00</td>
<td>0.88</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>9.84613E+04</td>
<td>2.431336306450874E+00</td>
<td>0.88</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>9.84614E+04</td>
<td>2.432035433504973E+00</td>
<td>0.88</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>9.84614E+04</td>
<td>2.432487552209051E+00</td>
<td>0.87</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>9.84615E+04</td>
<td>2.433181807201663E+00</td>
<td>0.87</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>9.84615E+04</td>
<td>2.433654624582289E+00</td>
<td>0.87</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>9.84616E+04</td>
<td>2.433945195628187E+00</td>
<td>0.87</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>9.84616E+04</td>
<td>2.434345197320533E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>9.84616E+04</td>
<td>2.434720053367289E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>9.84616E+04</td>
<td>2.434878081920510E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>9.84617E+04</td>
<td>2.435206048676719E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>9.84617E+04</td>
<td>2.435414672233243E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>9.84618E+04</td>
<td>2.435575730420423E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>9.84619E+04</td>
<td>2.4359590397176E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>9.84620E+04</td>
<td>2.436040181324504E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>9.84621E+04</td>
<td>2.436156310656338E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>9.84621E+04</td>
<td>2.436425423675610E+00</td>
<td>0.86</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>9.84622E+04</td>
<td>2.436577217314709E+00</td>
<td>0.86</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>9.84623E+04</td>
<td>2.436742372906690E+00</td>
<td>0.86</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>9.84624E+04</td>
<td>2.436962182406921E+00</td>
<td>0.85</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>9.84625E+04</td>
<td>2.43719579799458E+00</td>
<td>0.85</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>9.84625E+04</td>
<td>2.437183893264205E+00</td>
<td>0.85</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>9.84626E+04</td>
<td>2.437409109359719E+00</td>
<td>0.86</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>9.84626E+04</td>
<td>2.437434135390708E+00</td>
<td>0.86</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>9.84627E+04</td>
<td>2.437339333915531E+00</td>
<td>0.86</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NSTEP = 63  0.839928324099345E-05  6.41539051832934E-06  252.219  9.84618E+04  2.437333226307749E+01  0.86  0.15
NSTEP = 64  0.840112535171652E-05  6.417590060461500E-06  252.217  9.84618E+04  2.437208413858627E+01  0.86  0.15
NSTEP = 65  0.840301576936056E-05  6.42292888133574E-06  252.214  9.84618E+04  2.43687943853271E+01  0.86  0.16
NSTEP = 66  0.840496002461996E-05  6.42442686582981E-06  252.212  9.84618E+04  2.436703166375639E+01  0.86  0.16
NSTEP = 67  0.840692205541464E-05  6.4290712935445E-06  252.209  9.84618E+04  2.436408009242476E+01  0.86  0.16
NSTEP = 68  0.840882738940948E-05  6.431560468717111E-06  252.206  9.84617E+04  2.436031011269347E+01  0.86  0.16
NSTEP = 69  0.841071846400840E-05  6.434631620342549E-06  252.204  9.84617E+04  2.435683987643279E+01  0.86  0.16
NSTEP = 70  0.841247883599121E-05  6.436787931260484E-06  252.201  9.84617E+04  2.435327721859198E+01  0.87  0.16
NSTEP = 71  0.841410951797302E-05  6.438756780552661E-06  252.198  9.84616E+04  2.434924244743388E+01  0.87  0.16
NSTEP = 72  0.841570528993203E-05  6.440599213510488E-06  252.195  9.84616E+04  2.434545354776617E+01  0.87  0.16

*** HEADER FOR CCM3 HISTORY FILE ***

*** Primary History File ***

CASE:  clsst01
TITLE:  Test run: climatological SST

LENHDI MFTYP MFILE MFILEH NRBD MAXSIZ NDAVU MXXX NLON NOLON
292 43 2 20 3 33662 65282 0 128 128

PLAT PLEV PTRM PTRN PTRK NFLDH NSTEPH NSTPHH NITSLF NDBASE
64 18 42 42 42 85 72 0 72 0

NSBASE NNDCUR NSCUR NBDATE NBSEC NCDATE NCSEC MDT MHISF MFSTRT
0 1 0 000901 0 000902 1200 72 0

LENHDC LENHDR MPSIG MPLAT MPWTS MPFLDS MPFHL
259 239 1 112 176 38 90

MSS PATH NAME
CURRENT /JQUSER/csm/clsst01/ccm3/hist/h0001
FIRST /JQUSER/csm/clsst01/ccm3/hist/h0001
INITIAL /OLSON/datasets/T42/CCM3SEP1
TI BOUNDARY tibds.T42.0596
SST BOUNDARY STR-sst.T42.0596
OZONE BOUNDARY ozn.T42.0596
RGNFLS: Successfully opened file r0001
sending string to ishell from prealc:
setf -n 35580244 r0001.A
RGNFLS: Successfully opened file r0001.A
SAVDIS: shell version of mswrite is:

DATE TIME SEQ NO.
05/06/96 16:06:35 runmodel
05/06/96 16:04:02 runmodel
12/28/95 18:32:15 runmodel
12/27/91 09:12:22 - 1 -
09/21/92 13:34:26 on nrg= 8
on nrg1(nf)= 9
SAVDIS: shell version of mswrite is:
mswrite -nomail -nowait -t 100 -v CTPUBLIC -c "DAYS: 00000000.000-00000001.000 DATES: 0.000Z 000901 - 0.000Z 000902" - w passwd h000
1 /JQUSER/csm/clsst01/ccm3/hist/h0001

SAVDIS: Disposing Mass Store Volume/JQUSER/csm/clsst01/ccm3/hist/h0001
Write password = passwd
Retention Time = 100 DAYS
Cartridge = CTPUBLIC
Comment Field =
DAYS: 00000000.000-00000001.000 DATES: 0.000Z 000901 - 0.000Z 000902
Primary history file
Output at NSTEP = 73
Number of time samples on this file = 2
Model Day = 1.00

SAVDIS: shell version of mswrite is:
mswrite -nomail -nowait -t 100 -v CTPUBLIC -w passwd r0001 /JQUSER/csm/clsst01/ccm3/rest/r0001

SAVDIS: Disposing Mass Store Volume/JQUSER/csm/clsst01/ccm3/rest/r0001
Write password = passwd
Retention Time = 100 DAYS
Cartridge = CTPUBLIC

SAVDIS: shell version of mswrite is:
mswrite -nomail -nowait -t 100 -v CTPUBLIC -w passwd r0001.A /JQUSER/csm/clsst01/ccm3/rest/r0001.A

SAVDIS: Disposing Mass Store Volume/JQUSER/csm/clsst01/ccm3/rest/r0001.A
Write password = passwd
Retention Time = 100 DAYS
Cartridge = CTPUBLIC

SAVDIS: shell version of mswrite is:
mswrite -nomail -nowait -t 100 -v CTPUBLIC -w passwd rstrt /JQUSER/csm/clsst01/ccm3/rest/rstrt

SAVDIS: Disposing Mass Store Volume/JQUSER/csm/clsst01/ccm3/rest/rstrt
Write password = passwd
Retention Time = 100 DAYS
Cartridge = CTPUBLIC

Number of completed timesteps: 72
Time step: 73 partially done to provide convectively adjusted and time filtered values for history tape.
STOP executed at line 396 in Fortran routine 'CCM3'
CP: 549.471s, Wallclock: 174.650s, 39.3% of 8-CPU Machine
HWM mem: 10285307, HWM stack: 2056281, Stack overflows: 0
+ ja -sclhft

Job Accounting - Command Report

<table>
<thead>
<tr>
<th>Command</th>
<th>Started Elapsed</th>
<th>User CPU</th>
<th>Sys CPU</th>
<th>I/O Wait</th>
<th>I/O Wait</th>
<th>CPU MEM</th>
<th>I/O WMem</th>
<th>Kwords</th>
<th>Log I/O Memory</th>
<th>Ex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name At Seconds Seconds Seconds Sec Lck Sec Unlock Avg Mwds Avg Mwds Xferred Request HiWater St</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>--------</td>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>ja</td>
<td>16:00:25</td>
<td>0.2056</td>
<td>0.0005</td>
<td>0.0079</td>
<td>0.0641</td>
<td>0.1337</td>
<td>0.0773</td>
<td>0.0897</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>setf</td>
<td>16:04:20</td>
<td>0.1739</td>
<td>0.0076</td>
<td>0.0141</td>
<td>0.0703</td>
<td>0.0832</td>
<td>0.0473</td>
<td>0.0459</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ccm3bin</td>
<td>16:00:25</td>
<td>386.8616</td>
<td>549.4706</td>
<td>17.8048</td>
<td>5.8540</td>
<td>0.5163</td>
<td>2.7553</td>
<td>9.4415</td>
<td>10194.50</td>
<td>533</td>
</tr>
</tbody>
</table>

# 1 CPU  24.5493
# 2 CPU  52.2065
# 3 CPU  54.7026
# 4 CPU  71.9428
# 5 CPU  86.7592
# 6 CPU  122.9476
# 7 CPU  104.9368
# 8 CPU  31.4227
# SDS

2037504.00 65481

Job Accounting - Command Flow Report

parent ( CPU time) --> child ( CPU time) ...

ja  ( 0.0084)
ccm3bin  ( 567.2754)
setf  ( 0.0217)
setf ( 0.0196)

Job Accounting - Summary Report

Job Accounting File Name : /usr/tmp/nqs.+++++0gY/.jacct7902
Operating System : snl711 antero 8.0.4 har.8 CRAY Y-MP
User Name (ID) : jquser (xxxx)
Group Name (ID) : ncar (100)
Account Name (ID) : xxxxxxxx (xxxxx)
  eYhrs Allocated for Project : 12000.0000 (antero)
  eYhrs Used, as of 05/05/96 : 9981.4000
  eYhrs Allocated for Proposal : 35820.0000
  eYhrs Used, as of 05/05/96 : 24536.6000
Job Name (ID) : runmodel.init (7902)
Report Starts : 05/06/96 16:00:25
Report Ends : 05/06/96 16:06:51
Elapsed Time : 386 Seconds
User CPU Time : 549.4863 [ 549.4831] Seconds

Multitasking Breakdown

(Concurrent CPUs * Connect seconds = CPU seconds)

|ítulo | 1 * | 2 * | 3 * | 4 * | 5 * | 6 * | 7 * | 8 *
|------|-----|-----|-----|-----|-----|-----|-----|-----|

---

(System CPU Time : 17.8388 Seconds
Requested CPU Time : 1999.9784 Seconds
I/O Wait Time (Locked) : 6.0665 Seconds)
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Wait Time (Unlocked)</td>
<td>0.7342 Seconds</td>
</tr>
<tr>
<td>CPU Time Memory Integral</td>
<td>1563.0026 Mword-seconds</td>
</tr>
<tr>
<td>SDS Time Memory Integral</td>
<td>2415.7333 Mword-seconds</td>
</tr>
<tr>
<td>I/O Wait Time Memory Integral</td>
<td>55.2834 Mword-seconds</td>
</tr>
<tr>
<td>Data Transferred</td>
<td>9.9556 MWords</td>
</tr>
<tr>
<td>Maximum job memory used</td>
<td>10.1699 MWords</td>
</tr>
<tr>
<td>Maximum process memory used</td>
<td>9.8281 MWords</td>
</tr>
<tr>
<td>Requested job memory</td>
<td>11.0000 MWords</td>
</tr>
<tr>
<td>Maximum SDS used</td>
<td>14.0000 MWords</td>
</tr>
<tr>
<td>MPP Time</td>
<td>0.0000 PE-seconds</td>
</tr>
<tr>
<td>Logical I/O Requests</td>
<td>534</td>
</tr>
<tr>
<td>Physical I/O Requests</td>
<td>483</td>
</tr>
<tr>
<td>Number of Commands</td>
<td>4</td>
</tr>
<tr>
<td><strong>Charge Components</strong></td>
<td></td>
</tr>
<tr>
<td>CPU Hours</td>
<td>0.15759</td>
</tr>
<tr>
<td>CPU Charge</td>
<td>0.15759 eYhrs</td>
</tr>
<tr>
<td>Charge before Queue Factor</td>
<td></td>
</tr>
<tr>
<td>(Excluding MSS/NTWK/TAGS)</td>
<td></td>
</tr>
<tr>
<td>Multiplier for prem Queue</td>
<td>1.40</td>
</tr>
<tr>
<td>Charged against Allocation</td>
<td></td>
</tr>
<tr>
<td>+ exit 0</td>
<td></td>
</tr>
<tr>
<td>logout</td>
<td></td>
</tr>
</tbody>
</table>
Symbols

#if defined 151
#include 151

A
absems 108
absorptivity calculations 102
Absorptivity/Emissivity Arrays 108
Add Fields to History File, example 39
addfld.F 155
adding variables to Model 154
advnce.F 5, 100
after coupling 100
aphys.F 5, 102
array bounds checker 163
array dimensioning 104
assign command 16
asynchronous write 148
autotasking 115
Autotasking directives 116
AUXF 23, 61
Auxiliary History Files 61
averaging fields, history file 22

B
Base date 24
Base Model day 23
batch script 13
before coupling 100
bldfld.F 136
block data 113
BDTI 21, 30
BDTVS 21, 30
boundary datasets 46
branch run
  definition 27
  example 37
  .buffers 106

C
C preprocessor 3, 151
caldyi.F 100
Canopy water 49
case title 20
CCM Processor 45

CCM Web Page 93
CCM3
  compile and load 10
  conceptual flowchart 4
  design philosophy 95
  differences from CCM2 2
  Historical Development 1
  overview of structure 4
  printout 69
  standard mode, SST's 10
  supported architectures 10
ccm3.code.tar.gz 9
ccm3bin 12
CCMEXP 19
ccmoce.F 5
ccm-users e-mail group 93
cldint.F 5
cloud computations 102
cmfmca.F 5
Code Flow 99
  adjustment physics 102
  history file 136
  Initialization 99
  radiation 102
  restart and regeneration files 141
  semi-Lagrangian transport 100
  SLT constituent forecast 102
  Spectral Space 101
  tendency physics 102
  Time Integration (time-stepping) 99
Code Structure 167
Coding Standard 164
com3d 110
compiler directives 3
combslt 111
comspe 112
CONCHK 25
cond.F 5
costuent 103
constituents
  adjustment process 159
  global minima 159
  increase number of 156
  initial values 158
  initialize 157
  name of tracer 158
  naming 158
  save in history file 60
  surface fluxes, sources, sinks 158
Data Structures 103

datasets
  format 46, 138
  header record 138
  how to obtain from NCAR 9
  initial 45
  permanent storage at NCAR 12
  prescribed-value ozone 47
  Slab Ocean Model 49
  SOM time-varying boundary data 49
  time-invariant 47
  time-varying 47

date 24

debug 163
Debugging Run 43
Default File, Master Field List 52
Depends 151
Design Philosophy 95
deterministic results 115
DIAGNS, cpp token 152
DIF2 24
DIF4 24
diffusion calculations 101
diffusion coefficient 24
directory search path 145
Disk and Mass Store File Management 145
Distributed Memory
  I/O 119
  memory management 119
  multitasking 118
diurnal averaging 24
DIVDAMPN 24
DODIAGV 24
dry adiabatic adjustment 102
DTIME 24
dynamically allocate memory, history buffer 133
dyndrv.F 5, 101

e-mail 93
emissivity calculations 102
Index

writing fields to 137
writing to disk 137
history buffer regeneration file 141
history file
add field to 154
auxiliary 61
common blocks 138
content 51
data records 129
field type flag 22
format 121
header records 123
initialization 136
latitude data record 129
Mass Store archiving 67
message in printout 73
monthly-averaged 60
naming scheme 62
number of time samples 22
packing density 22
primary 52
printout of header 72
random order of latitude data records 97
write frequency 22
hordif.F 101
hstwr 137
hunit 137
hybrid sigma-pressure coordinate 103
HYDRO 25

I
I/O in the multitasked code 117
included files, #include 3
INCORBUF 25
In-Core Grid-point Arrays 110
in-core memory 103
INCORHST 25
INCORRAD 25
indefinite, initializing as 163
inidat.F 99
initial.F 5, 99
initcom.F 5
initext.F 5
Initial Dataset 45
Initial Run
definition 27
Generate a Monthly-Averaged History File 42
Run with Multiyear SST (AMIP) Dataset 40
T21 example on a SUN or RS-6000 32
T42 example on NCAR Cray 29
initialization phase 4
initialization run 4
see initial run
Input Datasets 45
instantaneous fields, history file 22
inthed.F 137
intht.F 5, 133, 136
inti.F 5

IRADAE 24, 102
IRADLW 24, 102
IRADSW 24
IRT 21
ishell 149
ITSST 24

J
ja command 74
Job accounting 74

K
KMXHDC 24
KPT, cpp token 154

L
Land Surface Model
boundary dataset 48
initial dataset 49
parallel processes 70
Land Surface Model (LSM) datasets 48
large-scale condensation 102
lat 103
Latitude Data Records 129
Latitude indices 103
LCROOT 21
leapfrog time-stepping scheme 103
Legendre polynomials 112
Level indices 103
lbuf 132
library routines 15
linemsac.F 102
linsmbc.F 102
Local Workspace 109
logical units
code flow 99
printout of assignments 71
Longitude indices 103
LPT, cpp token 154
LSM
see Land Surface Model
LSMEXP 25
lsmi.F 5, 100
lsmini.F 5
LSMLAT, cpp token 153
LSMLON, cpp token 153
lunits.F 99

M
main model buffer 103, 106
makdep 151
make 3, 10, 151
Makefile 3
Mass Storage System 149
Mass Store


archiving 67
Retention period 21
retention times 67
Virtual volume 21
write password 21
massively-parallel 115
Master Field List 52
add field to 154
master regeneration file 140
Master regeneration filename 65
maximum values, history files 22
MCFLDS 129
memory management goals of CCM3 95
memory statistics 71
Memory usage in multitasked code 118
meridional wind component 46
Message Passing
distributed memory 118
message passing
documentation 10
Parallel Virtual Machine 16
MFILT 22, 52
MFLDS 129
mgwintr.F 5
mid-month regeneration files 66
minimum values, history file 22
msksic.F 138
Model xiii
moist convection 102
MRESFQ 23, 63
msread 149
mswrite 148
multitasking
directives 3
distributed memory 118
in LSM 70
message passing 118
multitasked loops 116
shared memory 115
Multitasking Strategy 115
multiyear time-varying boundary dataset 40, 47
MXSZRG 23

N
n3 101, 111
n3ml 101, 111
nabem 108
namelist
CCMEXP 19
files nl.init and nl.restart 19
general description 19
LSMEXP 25
variables that must be set 25
namelist variables
Example Runs 28
History file options 22
Input datasets 21
LSM 25

Mass Store controls 21
Memory controls 25
Model time 23
Physics controls 24
presetting 99
Restart/Regeneration file options 23
type and length of run 20

naming files
history files 62
regeneration files 64
restart file 64

navu.F 137
NCDATA 21, 30
NCPUS 12, 115
NDENS 22, 30, 52
NELAPSE 20
NESTEP 20, 31
netlib 10
Network Queuing System 13
New Variables 154
NHTFRQ 22, 31, 52
NINAVG 22, 52
NLFILT 22
NLVDRY 24
NNBDAT 24
NNBSEC 24
NNDBAS 23
NNSBAS 23
NPES 17
NQS 13
nra1 108
nrbl 108
NREFRQ 23, 63
NREVSN 23
NRMVN 21
NSMVN 21
NSREST 20
NSVSN 23
NSWRPS 21
NUMULV, cpp token 154
nxpt 104

O
object file 4
Objects 151
ocean mixed layer depth 49
ocean mixed layer heat flux 49
ORO 45
orography flag 50
outfld.F 155
Out-of-Core Data Storage 108
out-of-core memory 103
output dataset naming
history files 62
regeneration files 64
restart file 64
OZNCYC 21
ozone volume mixing ratios 47
P

parallel processing
see multitasking
parallel tasks, LSM 70
parameterization
adding to Model 160
common blocks 113
initialization routines 112
interface 112
interface routines 113
pcnst 103
PCNST, cpp token 153
PERGRO 25
perl 151
PERTLIM 24
pflds 20
PHIS 45
phis (surface geopotential) 50
physical parameterizations
adding to Model 160
parameterization interface 112
physics
adjustment 5, 102
tendencies 5, 102
plat 105, 121
PLAT, cpp token 153
platd 105
PLENR, cpp token 153
PLEV, cpp token 153
plevd 111
plon 104
PLON, cpp token 153
plond 104
Plug-compatible physics routine 96, 112
pointer variables 99, 106
points.F 99
preallocate disk space 148
preprocessor directives 3
Prescribed-Value Dataset, ozone 47
preset.F 5, 99
PRIMARY 22, 52
Primary History File 52
primary regeneration file 140
Printout, CCM3 run 69
prng1 140
prng2 140
prng3 140
ptapes 20
PTRK, cpp token 153
PTRM, cpp token 153
PTRN, cpp token 153
PVM 16
PVM environment resource variables 18
PVME, cpp token 152
PVP 10
Q

QSUB directives 15
quad.F 101
R

radclw.F 102
radctl.F 5, 102
radiation 102
radiation buffer 103
rdharr.F 138
RDLSF 25
readrg.F 142
Regeneration Datasets 63
regeneration files 138
common blocks 143
history buffer 141
initialization 141.
Mass Store archiving 67
master 140
mid-month 66
naming scheme 64
primary 140
reading 141
secondary 141
write frequency 63
writing 142
Regeneration Run

definition 27
eexample 36
resolution, changing 160
Resource Allocation Errors 161
restart file 63, 138
contents 140
write frequency 63
writing 142
Restart filename 65
Restart Run

definition 27
eexample 34
restrt.F 141
resume.F 5
rgnhbf.F 137
RIRT 21
Routine Names 167
RS6K, cpp token 10, 152
S

savdis.F 138
scan1ac.F 100
scan1bc.F 5, 100
scan2.F 5, 101
scanslt.F 5, 100
sea ice thickness 49
search path 145
sea-surface temperature 47
Secondary Data Segment 16
secondary regeneration file 141
temporal disk space 148
Time filter coefficient 24
time sample 51, 121
time, of simulation 100
Time-Invariant Boundary Dataset 47
timestep 24
time-stepping phase 4
Time-Variant Ozone, Prescribed-Value Dataset 47
Time-Variant Sea-Surface Temperature (SST) Dataset 47
topography standard deviation field 47
tphysac.F 5, 102
tphysbc.F 5, 102
trace constituents, history file 60
tracer name 158
Trouble-Shooting 161
tstep.F 101

U
unblocked files 108
uncompress file (gunzip) 9
UNICOS operating system 14

V
Variable Names 167
vdintr.F 5
vectorization 3
Vegetation temperature 49
vertical coordinate 103
Volumetric soil water content 49

W
water vapor specific humidity 46
whenwr.F 137
work file 108
World Wide Web 9
wrtrsl.F 5, 141
writing files
history files 22, 52, 137
message in printout 73
regeneration files 23, 63, 142
restart 63, 142
restart file 23
wrtrsl.F 5, 137
wsds.F 140

Z
zonal wind component 46