

**Community Sea Ice Model (CSIM) Developer's Guide  
Code Reference for Version 5.0**

Released with CCSM3.0

Julie Schramm  
Cecilia Bitz  
Bruce Briegleb  
Marika Holland  
Elizabeth Hunke  
Bill Lipscomb  
Dick Moritz

**Community Climate System Model**

National Center for Atmospheric Research, Boulder, CO

<http://www.cgd.ucar.edu/csm/models>

---

CVS tag `$$Name$` Build date: June 4, 2004

# Contents

<b>1 Overview</b>	<b>2</b>
1.1 CCSM Directory Structure . . . . .	2
1.2 CSIM Module Descriptions . . . . .	3
1.3 External Software and Libraries . . . . .	4
1.3.1 CCSM Share Code . . . . .	4
1.3.2 netCDF Library . . . . .	5
1.3.3 Message Passing Interface (MPI) . . . . .	5
<b>2 Model Structure</b>	<b>5</b>
2.1 Time-stepping Loop . . . . .	5
2.2 Information Exchanged with the Coupler . . . . .	6
2.3 Global Grid and Grid Decomposition . . . . .	6
<b>3 Making Code Modifications</b>	<b>9</b>
3.1 Write Statements and Error Messages . . . . .	9
3.2 History Fields . . . . .	9
3.3 Restart Fields . . . . .	11
<b>4 Code Management under CVS</b>	<b>11</b>
<b>5 Coding Standard</b>	<b>11</b>
5.1 Style Guidelines . . . . .	11
5.2 Content Guidelines . . . . .	15
<b>6 Software Integration Procedure</b>	<b>15</b>
6.1 Making a Control Run . . . . .	16
6.2 Testing Climate and Performance of Modified Code . . . . .	16
<b>7 Glossary</b>	<b>17</b>
<b>References</b>	<b>18</b>
<b>A Calling Tree</b>	<b>18</b>
<b>B ProTeX Generated Documentation</b>	<b>20</b>



To make modifications to the coupled model setup, modify the scripts in **ccsm3/scripts**. The setup scripts for the uncoupled model are located in **ccsm3/models/ice/csim4/src**. The files located under the main ice model directory **/csim4/src** are described in the next section.

## 1.2 CSIM Module Descriptions

The ice model uses **ProTex**, which is a tool for self-documenting source code. ProTex is a perl script that produces a LaTeX compatible document from Fortran source code that contains a ProTex prologue at the top of each module, function and subroutine. The output from ProTex can be over 100 pages, and make files and build scripts would not be included in any documentation created by the ProTex tool. In light of this, the ice model source code and the build scripts are described briefly in this section. There is additional information in the code at the top of each module, subroutine and function.

**csim4/src/csim.setup.csh** compiles the uncoupled ice model and prestages the input data, contains the namelist, used by **csim\_run**

**csim4/src/csim\_run** runs the uncoupled ice model

**csim4/src/IBM\_tests** runs debug and exact restart tests on the **NCAR** IBM for the uncoupled ice model

**csim4/src/bld/Macros.OS** macro definitions for supported operating systems, used by **Makefile**

**csim4/src/bld/makdep.c** perl script that determines module dependencies

**csim4/src/bld/Makefile** builds the uncoupled ice model

**csim4/src/input\_templates/ice\_model\_size.F.\*** input files prestaged by **csim.setup.csh** depending on the grid dimensions and number of ice thickness categories set in this file.

Below is a list of one-line descriptions for the ice model modules from **csim4/src/source**.

**ice.F** main program

**ice\_albedo.F** albedo parameterization

**ice\_atmo.F** stability-based parameterization for calculating turbulent ice-atmosphere fluxes

**ice\_calendar.F** time manager

**ice\_constants.F** parameters, physical and numerical constants

**ice\_coupling.F** interface with flux coupler

**ice\_diagnostics.F** diagnostic and debugging routines

**ice\_domain.F** **MPI** subdomain sizes and parallel processing info

**ice\_dyn\_evp.F** elastic-viscous-plastic dynamics component

**ice\_exit.F** aborts the model and prints error message

**ice\_fileunits.F** unit numbers for I/O

**ice\_flux.F** fluxes needed/produced by the model

**ice\_flux\_in.F** routines to read and interpolate forcing for stand-alone ice model

**ice\_grid.F** grids and land masks

**ice\_history.F** netCDF output routines and read/write restart

**ice\_init.F** namelist and initializations

**ice\_itd.F** utilities for managing ice thickness distribution

**ice\_itd\_linear.F** linear remapping for transport in thickness space

**ice\_kinds\_mod.F** definitions of reals, integers, etc.

**ice\_mechred.F** mechanical redistribution (ridging)

**ice\_model\_size.F** grid size, number of thickness categories and vertical layers

**ice\_mpi\_internal.F** utilities for internal MPI parallelization

**ice\_ocean.F** slab ocean mixed layer model

**ice\_prescribed.F** prescribed ice (not supported in this release)

**ice\_read\_write.F** utilities for reading/writing files

**ice\_scaling.F** ice-area scaling of fluxes sent to coupler

**ice\_state.F** arrays defining ice state

**ice\_therm\_itd.F** thermodynamics changes mostly due to ITD (post-coupling)

**ice\_therm\_vertical.F** vertical growth rates and fluxes (pre-coupling thermodynamics)

**ice\_timers.F** timing routines

**ice\_transport\_mpdata.F** horizontal advection via MPDATA or upwind

**ice\_transport\_remap.F** horizontal advection via incremental remapping

**ice\_work.F** globally accessible work arrays

### 1.3 External Software and Libraries

The CSIM code includes references to subroutines and libraries not contained in **models/ice/csim4**. This code is described in the following sections.

#### 1.3.1 CCSM Share Code

CSIM uses some of the CCSM shared-code modules that are included in the **models/csm\_share** directory. The share-code modules presently used in CSIM are:

- **shr\_const\_mod.F90** - This module contains the "shared constants" that are used by all components for CCSM. Note that these constants are not used for the uncoupled ice model, which uses those set in **ice\_constants.F**.
- **shr\_kind\_mod.F90** - This module defines variable precision used by other shared-code modules.
- **shr\_msg\_mod.F90** - This contains subroutines that provide an architecture-independent means of interacting with the operating system. CSIM uses the following subroutines from this module:
  - *shr\_msg\_chdir* - changes current working directory
  - *shr\_msg\_dirio* - redirects standard input and output to named files
- **shr\_mpi\_mod.F90** - used by **shr\_sys\_mod.F90**
- **shr\_sys\_mod.F90** - This contains subroutines that provide an architecture-independent means of interacting with the operating system. CSIM uses the following routines from this module:
  - *shr\_sys\_flush* - clears the print buffer
  - *shr\_sys\_abort* - provides a consistent stopping mechanism for multiple processors

### 1.3.2 netCDF Library

The netCDF library is necessary to run CSIM. The slab ocean mixed layer model and the prescribed ice model (not supported in this release) read in forcing data from netCDF files. The output history files are also written in this format. The history files conform to the NetCDF Climate and Forecast (CF) Metadata Conventions (<http://www.cgd.ucar.edu/cms/eaton/cf-metadata/index.html>). All of the netCDF-specific code used to write the history files is in subroutine *icecdf* in module **ice\_history.F**. Machine dependent information that sets the location of the netCDF library is set in the **models/bld/Macros.\*** for the coupled model, and in **models/ice/csim4/src/bld/Macros.\*** for the uncoupled ice model.

### 1.3.3 Message Passing Interface (MPI)

CSIM intra- and inter-model communications are conducted via MPI. The MPI library is automatically loaded during the CCSM build procedures.

## 2 Model Structure

CSIM5 is written in the Fortran 90 programming language in fixed source form. It is parallelized internally by grid decomposition using **MPI** (Message Passing Interface). There is an external layer of parallelization, also using MPI, that allows message passing between the ice model and the coupler.

### 2.1 Time-stepping Loop

The main program, **ice.F**, contains the time-stepping loop. The model flow at the highest level is shown below.

```
ICEMODEL--initialize variables
|
| do time-stepping loop:
| |                                     +-----<-----+
| |                                     | +---<---+ | | |
| | +-receive info from coupler        | v         | |
| |                                     |fsalt_hist=0| |
| | +-vertical thermodynamics calculations | |         | |
| |                                     | v         | |
| | +-send ice state and fluxes to coupler | |         | |
| |                                     | |         | |
| | +-thickness redistribution thermodynamics | |         | |
| |                                     fsalt=0 |         ^ ^
| | +-dynamics calculations             | |         | |
| |                                     v         | |
| | +-write std out to log file          | |         | |
| |                                     | |         | |
| | +-write avg quantities to history file | +--->---+ |
| |                                     +----->-----+
|
| end time-stepping loop
|
+-exit_coupler
```

A complete calling tree is shown in Appendix A. The ice model receives information from the coupler at the beginning of the time loop. The vertical ice thermodynamics are calculated, and the ice state and fluxes are returned to the coupler at mid-timestep immediately after they are computed. The information is sent to the coupler at this point in the time-stepping loop for **load balancing**. This allows the atmosphere to begin calculations instead of waiting until the end of the ice model timestep. The thermodynamics calculated after

returning data to the coupler are mostly related to the ice thickness distribution and include transport in thickness space, lateral growth and melting, and freeboard adjustment.

### **hist Variables**

As shown in the model flow in Section 2.1, fluxes are sent to the coupler in the middle of the timestep, and the diagnostics are written out at the end of the timestep. Most fluxes, like the atmospheric latent heat, are initialized at the beginning of the timestep and calculated only in the thermodynamics. This is the value that is sent to the coupler and then added to the average being calculated for the history file.

A problem arises for fluxes that are modified before and after the fluxes are sent to the coupler, that is, in both the thermodynamics and dynamics calculations. These fluxes are `fsalt`, `fresh`, `fhnet`, and `fwthru`. If these values are initialized at the start of the time step, the values computed in the dynamics during the previous time step would be written over. The fluxes sent to the coupler would not represent quantities summed over an entire timestep. This is why these fluxes are initialized immediately after the call to the coupler.

If these fluxes (`fsalt`, `fresh`, `fhnet`, `fwthru`) were written to the history file, the fluxes computed at the beginning of the step (before the call to the coupler) would be neglected. The variables with `hist` appended to the name are initialized before the thermodynamics calculation and measure the same thing as their coupler counterparts but over different time intervals. The `hist` variables contain information calculated from the start to the end of a timestep and are written to the history file.

To calculate the global sums for the diagnostics, there is a value of the ice concentration `aice_init` that is saved from the beginning of the timestep.

## **2.2 Information Exchanged with the Coupler**

When CSIM5 is run coupled, it sends and receives forcing information from the other components via a flux coupler. Message passing between the ice model and the coupler is accomplished using MPI. Fluxes computed within the ice model and used by other components are sent to the flux coupler for distribution. Although CSIM5 contains an ice thickness distribution in each grid cell, and the ice state variables and fluxes depend on the ice thickness, only aggregate quantities of each grid cell are passed to the coupler.

The coupler requires that the fluxes it receives from the ice model be divided by the total ice area in each grid cell, since the coupler multiplies these fluxes by the ice area. This is done in subroutine `scale_fluxes` just before the call to `to_coupler`. These fluxes have units of "per unit ice area".

The forcing information received by the ice model from the coupler at the top of the timestep is listed in Table 1. By convention, directional fluxes are positive downward. The symbols in the first column correspond to those in the equations found in the Scientific Description document. These are the forcing variables required by the ice model for running coupled or uncoupled. The information calculated by the ice model and sent to the coupler at mid-timestep is listed in Table 2.

## **2.3 Global Grid and Grid Decomposition**

CSIM uses a generalized orthogonal B-grid, where tracer quantities are located at the center of the grid cells, and velocities are located at the corners. The internal ice stress tensor takes four different values within a grid cell. Tracer quantities in the ice model include ice and snow area, volume, energy and temperature. The grid comprised of the center points of the grid cells is referred to as the "T grid". The "U grid" is comprised of the points at the northeast corner of the corresponding cell on the T grid. Quantities that are defined on the U grid include ice and ocean dynamics variables.

To achieve better performance on cache and vector machines, the global domain is separated into subdomains. Two criteria should be kept in mind when choosing the size of the subdomains:

1. The number of subdomains should divide evenly into the global domain in both directions.
2. The global grid should be divided so that there is ice in each subdomain. This will divide the work more evenly between the processors.

Table 1: Fluxes and state variables received by the sea ice model from the coupler

Symbol	Variable Name	Description	Units
Atmospheric Variables			
$z_a$	zlvl	Reference height	m
$u_a$	uatm	Zonal wind speed at $z_a$	$\text{m s}^{-1}$
$v_a$	vatm	Meridional wind speed at $z_a$	$\text{m s}^{-1}$
$\theta_a$	potT	Potential temperature at $z_a$	K
$T_a$	Tair	Air temperature at $z_a$	K
$q_a$	Qa	Specific humidity at $z_a$	$\text{kg kg}^{-1}$
$\rho_a$	rhoa	Air density at $z_a$	$\text{kg m}^{-3}$
Atmosphere $\Rightarrow$ ice fluxes			
$F_{SWvdr}$	swvdr	Direct, visible downwelling shortwave	$\text{W m}^{-2}$
$F_{SWvdf}$	swvdf	Diffuse, visible downwelling shortwave	$\text{W m}^{-2}$
$F_{SWndr}$	swidr	Direct, near infrared downwelling shortwave	$\text{W m}^{-2}$
$F_{SWndf}$	swidf	Diffuse, near infrared downwelling shortwave	$\text{W m}^{-2}$
$F_{LWDN}$	flw	Downwelling longwave	$\text{W m}^{-2}$
$F_{RN}$	frain	Freshwater flux due to rain	$\text{kg m}^{-2} \text{s}^{-1}$
$F_{SNW}$	fsnow	Freshwater flux due to snow (liquid)	$\text{kg m}^{-2} \text{s}^{-1}$
Ocean Variables			
$T_o$	sst	Sea surface temperature	K
$S_o$	sss	Sea surface salinity	ppt
$u_o$	uocn	Surface ocean current	$\text{m s}^{-1}$
$v_o$	vocn	Surface ocean current	$\text{m s}^{-1}$
$H_{ox}$	ss_tltx	Sea surface slope	$\text{m m}^{-1}$
$H_{oy}$	ss_tlty	Sea surface slope	$\text{m m}^{-1}$
Ocean $\Rightarrow$ ice fluxes			
$F_{Qoi}$	frzmlt	Freezing/melting potential	$\text{W m}^{-2}$

Table 2: Fluxes and state variables sent from sea ice model to coupler

Symbol	Variable Name	Description	Units
$T_{ref}$	<b>Tref</b>	Atmospheric reference temperature (2 m)	K
$Q_{ref}$	<b>Qref</b>	Atmospheric specific humidity (2 m)	kg kg <sup>-1</sup>
Ice Variables			
$A$	<b>ailohi</b>	Ice concentration	
$T_s$	<b>Tsfc</b>	Surface temperature	K
$\alpha_{vdr}$	<b>alvdr</b>	Albedo (visible, direct)	
$\alpha_{ndr}$	<b>alidr</b>	Albedo (near infrared, direct)	
$\alpha_{vdf}$	<b>alvdf</b>	Albedo (visible, diffuse)	
$\alpha_{ndf}$	<b>alidf</b>	Albedo (near infrared, diffuse)	
Ice $\Rightarrow$ atmosphere fluxes			
$F_{LH}$	<b>flat</b>	Latent heat flux	W m <sup>-2</sup>
$F_{SH}$	<b>fsens</b>	Sensible heat flux	W m <sup>-2</sup>
$F_{LWUP}$	<b>flwout</b>	Upwelling longwave	W m <sup>-2</sup>
$F_{EVAP}$	<b>evap</b>	Evaporated water	kg m <sup>-2</sup> s <sup>-1</sup>
$\tau_{ax}$	<b>tauxa</b>	Atmosphere-ice stress, zonal	N m <sup>-2</sup>
$\tau_{ay}$	<b>tauya</b>	Atmosphere-ice stress, meridional	N m <sup>-2</sup>
Ice $\Rightarrow$ ocean fluxes			
$F_{SWo}$	<b>fswthru</b>	Shortwave transmitted to ocean	W m <sup>-2</sup>
$F_{Qio}$	<b>fhnet</b>	Net heat flux to ocean	W m <sup>-2</sup>
$F_{Wo}$	<b>fresh</b>	Fresh water flux	kg m <sup>-2</sup> s <sup>-1</sup>
$F_{Sa}$	<b>fsalt</b>	Salt flux	kg m <sup>-2</sup> s <sup>-1</sup>
$\tau_{ox}$	<b>tauxo</b>	Ice-ocean stress, zonal	N m <sup>-2</sup>
$\tau_{oy}$	<b>tauyo</b>	Ice-ocean stress, meridional	N m <sup>-2</sup>

There are NX and NY subdomains in the  $x$  and  $y$  directions, respectively. NX and NY are set in **csim\_run** for the uncoupled ice model and are set automatically in the CCSM scripts. These values are used in the **Macros.\*** files as C pre-processor flags.

The dimensions of the global domain are **imt\_global** x **jmt\_global**, and those of the subdomains are **imt\_local** x **jmt\_local**. The physical portion of a subdomain is dimensioned as [**ilo:ihi,jlo:jhi**], with **num\_ghost\_cells** around the outside of the physical domain for boundary conditions.

Figure 1 shows a schematic of the grid decomposition for the gx3 grid divided into 4 processors in the  $x$  direction and 2 in the  $y$  direction. Note that the first processor is in the lower left corner and is numbered zero. An exploded view of a subdomain is shown. The values of **imt\_local** and **jmt\_local** include the ghost cells.

Typically, when the ice model stops due to a conservation error or a CFL violation, the coordinates of the local subdomain and the processor number are printed out, not the global coordinates. The conversion from local coordinates on a given processor to global coordinates is printed out in the log file. It gives the local array size, and the global coordinate start for each processor. Shown below is the output for the example shown in Figure 1.

Document Grid and Subdomain Sizes:

=====

```

Global problem size:      100 x   116
Using      8 processors in a    4 x       2 Cartesian decomposition
Local array size is:      27 x   60
Physical domain is (approximately):  25 x   58
Local i,j start for each processor:    2     2
Local i,j end   for each processor:    26    59

```

```
Global i start for each processor:  1 26 51 76  1 26 51 76
Global j start for each processor:  1  1  1  1 59 59 59 59
```

This example is from the gx3v5 grid with `imt_global x jmt_global = 100 x 116`. The local array size is `imt_local x jmt_local = 27 x 60`, including ghost cells. The physical domain is `[ihi-ilo+1, jhi-jlo+1] = [25, 58]`; this does not include the ghost cells. Each physical subdomain starts at `[ilo = 2, jlo = 2]` and ends at `[ihi = 26, jhi = 59]`. These are the loop indices for most do loops in the model. The last two rows are the global indices for the southwest point of each subdomain. These are useful for converting local indices to correspond to the global indices in the history file, for example.

### 3 Making Code Modifications

The source code for CSIM is located in `ccsm3/models/ice/csim4/src/source`.

NOTE: The source code in this directory, and any code that has been checked out of the CVS repository should be treated as frozen code. It is recommended that the permissions on these files be changed to read only. To modify a module in CSIM, whether running coupled or uncoupled, first copy that module to a separate directory. If running CSIM coupled, this directory is near the top of the CCSM directory tree and is called `ccsm3/scripts/$YOUR_CASE/SourceMods/src.csim`. If running CSIM uncoupled, make a directory under the active ice component called `models/ice/csim4/src/src.ice`. Make the modifications to the copied file. The scripts are set up so that these directories are the last in the filepath, so the modified files are the last copied to the executable directory. This is a highly recommended programming practice, keeping your modifications separate from the original code.

#### 3.1 Write Statements and Error Messages

Adding write statements to source code that is using multiple processors can produce unexpected results if not done correctly. Generally, diagnostic write statements should be done only by the controlling processor, called the `master_task`. The task number for the master processor is always zero and is set in `setup_mpi`. The master task is the only processor that can write to the log file. If other tasks are allowed to write to the log file, output from the master task will most likely be overwritten. Write statements should be surrounded by an 'if' block:

```
if (my_task == master_task) then
  write (nu_diag,*) 'istep1:',istep1
endif
```

Without the 'if' statement, all processors will write out the information. Output from other processors will be written to the standard output file.

#### Finding Error Messages

Errors that cause the ice model to stop do not always occur in the master task subdomain, so there may not be information about the error in the log file. When the model stops, look for error messages in the log files, standard output, standard error, `fort.*`, and core files. Since multiple processors are writing to the standard output file, the error that caused the model to stop is not always right at the bottom of the file.

#### 3.2 History Fields

Adding a field to the history namelist is different than adding a field to the history file. If the field is already in the namelist, it can be added or removed from the history file simply by modifying the namelist called `ice_fields_nml`. This is discussed in the CSIM User's Guide.

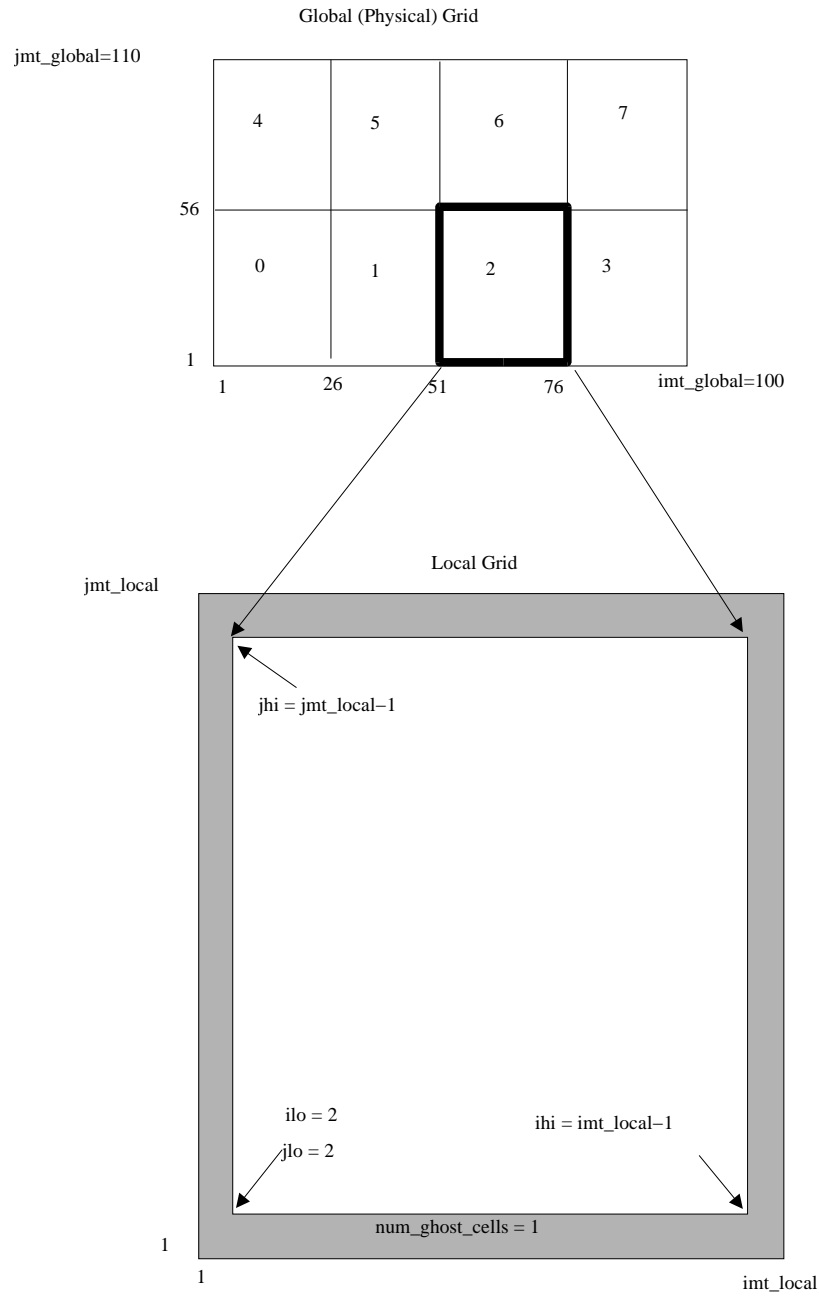


Figure 1: An example of the gx3 (imt\_global=100, jmt\_global=116) grid with the following decomposition: 4 processors in the x direction, 2 in the y direction. Grey shading represents ghost cells that are filled in by a call to *bound* with redundant data from neighboring processes.

The fields that are available to the history file namelist are set in **ice\_history.F**. At the top of this module are instructions on what to modify there, in the namelist in the ice setup script, and in subroutines *init\_hist* and *ice\_write\_hist*.

### 3.3 Restart Fields

Fields that are necessary for an exact restart should be added to the restart file. These are binary files that are read by the ice model for initial conditions for almost all types of runs. There are two subroutines in **ice\_history.F** called *restartfile* and *dumpfile* that read and write these files. The actual reading of the unformatted file is done by subroutine *ice\_read* in module **ice\_read\_write.F**. New fields should be added to the end of the restart file to allow backwards compatibility, i.e. older versions of the model should be able to read the new restart file. There is code in *restartfile* that checks for the end of the file. Currently, the salt flux is the last field in the file. If this field is present, it is read in, if not, it is set to zero. An exact restart test should be done after modifying the restart file. This test is automated in the CSIM and CCSM scripts. Adding more fields to the restart file should only be done if required by physics changes. It needs to be done carefully, considering the need for backwards compatibility.

## 4 Code Management under CVS

CSIM uses **CVS** for revision control. CVS is a tool that documents the history of source code changes and maintains this history on a source tree. It documents each change with a time stamp and the user name of the person who made it. The user making the changes adds some text describing the changes. A tag is created, which gives a label to the collection of revisions.

Tags on the **CVS main trunk**, the main line of development for CSIM, are of the form *csim#-#-#*, where # represents an integer. Modifications to the first number are for major model versions that usually coincide with CCSM releases. Increments are made to the middle number when model physics are significantly modified or added. These versions are typically not publically released. Changes to the last number represent incremental changes to the model, such as bug fixes or cosmetic changes.

Development work that is experimental or requires major code changes should be done on a **CVS branch**, a line of development separate from the main trunk. Branches are used so that the changes do not appear on the main trunk until they are thoroughly tested. CVS branch names are of the form *csim#-#-#.brnch\_desc* where the first three numbers denote the tag on the main trunk where the branch was created. The string of characters at the end of the tag name gives the purpose of the branch. For example, *csim4.8.16\_brnch\_bugfix* was rooted at version *csim4.8.16* on the main trunk, and is a branch to make a bug fix. Tags created along the branch should have the form *csim#-#-#.brnchT\_desc#*. This is basically the branch name with a 'T' to differentiate the tag name from the branch name, and a number at the end to denote where along the branch the tag was made.

## 5 Coding Standard

The goal of a coding standard is to create source code that follows a consistent style, giving the sea ice model a common look and better readability. For historical reasons, the current code does not completely conform to all of the following standards. Work is in progress to correct this. The coding standard will be enforced through code reviews.

### 5.1 Style Guidelines

#### General Guidelines

- **Fortran 90 Standard** CSIM uses the F90 standard.
- **Preprocessor** The C preprocessor is required as the macro processor (cpp). All tokens should be uppercase to distinguish them from fortran code. The use of *ifdef* constructs should be avoided wherever possible. The option should be set in the namelist instead.

- **Fixed-Form Source** Fixed-form source will be used for readability. Columns 7 to 72 of a line may contain a Fortran statement, while columns 1 to 6 are reserved for special purposes. Blanks may be used freely anywhere. Exclamation marks should be used to denote comments. A statement may include a maximum of 19 continuation lines.
- **Bounds Checking** All code should be able to run when the following compiler options are set:
  - Array bounds checking
  - Automatic variables are initialized to NaN
  - Floating point exception conditions: overflow, division by zero, and invalid operations

## Modules

- The use of modules is strongly encouraged since they provide global access to variables.
- Modules must have the same name as the file in which they reside, due to dependency rules used by "make" programs. This implies that multiple modules are not allowed in a single file.
- Module names must begin with "ice\_". This will provide unique module names when CCSM is released as a single executable version.
- Related subroutines and variables should be placed in a single module/file rather than keeping a single routine per file.
- The use of common blocks is discouraged. Modules are a better way to provide access to declared entities via the "use" statement.
- If possible, access to variables should be limited through use of **only** and **private** clauses in the "use" statements.

## Subroutines

- **Inline** any subroutine or function that is only called once, unless it contains a significant amount of physics.
- Minimize work arrays and other extraneous copies unless the array is used multiple times within the same subroutine. Globally accessible work arrays are available in **ice\_work.F**.

## Loops

- Loops should be structured with the **do-endo** construct as opposed to numbered loops.
- Loops should be combined as much as possible for readability and performance.
- Subroutine calls within loops over horizontal indices *i* and *j* should be avoided to produce a more vector-friendly code.
- Short loops should be placed outside of longer loops to produce vector-friendly code. For example, loops over ice thickness categories should be placed outside of loops over horizontal indices *i* and *j*.

## Array Syntax

- Whole array expressions are encouraged for performance reasons. For example,

```
do i = 1, imt
  a(i) = b(i) + c(i)
endo
```

is more desirable than

$$a = b + c$$

## Allocatable Arrays

- Allocatable arrays should be avoided for performance reasons.

## Variable Names

- **State Variables** Primary state variables should use the following naming convention:

- `aicen(i,j,n)` ice concentration for each ice category
- `aice(i,j)` ice concentration aggregated over all ice categories
- `ain(n)` ice concentration in a single column, for each ice category

Variables that are not state variables but are used in a similar manner should also follow this naming convention. Note that `ai` is not a good choice for a variable name.

- **Descriptive Names** Variable names should be descriptive and long enough to be found, for example with the UNIX "grep" utility, without pages of extraneous material. For example, `eice` and `esno` are preferable to `ei` and `es`.
- **Variables with Similar Names** Variables with the same name, but with different numbers of arguments should not be used in different places. For example, an array `Ts(i,j)` in one subroutine should not be used as a local variable `Ts` in another. An exception to this would be the `work` arrays, which should always be local.

## Variable Declarations

- Variables should be declared in F90 format, using a double colon.
- The F90 `kind` type should be used for all variable declarations.
- Continuation lines are encouraged for declarations of arrays that have similar kinds and dimensions. For example:

```
real (kind=dbl_kind), dimension (imt_local,jmt_local) ::
&  aice      ! concentration of ice
&,  vice     ! volume per unit area of ice           (m)
&,  vsno     ! volume per unit area of snow          (m)
```

- Variables should be declared one per line to allow a comment field after it, with a "!" character followed by the comment text.
- Variables of a similar type and function may be grouped together on a single line. For example:

```
integer (kind=int_kind) :: nyrp,mdayp,weekp      ! previous year, day, week
```

## Code indentation

- Code within loops and if blocks will be indented three characters.
- Continuation lines of an assignment statement should begin to the right of the assignment operator. For example,

```
    Fresh(i,j) = Fresh(i,j) + (vsn(nc)*Rside*rhos
$                + vin(nc)*Rside*rhoi)/dt
```

- Continuation lines of a subroutine call should begin to the right of the "(" character. For example,

```
    call comp_matrices ( rowflg, hin, Hmean,
$                      Gamm, HGamm, H2Gamm )
```

## Commenting of code

- Comments enclosed in horizontal lines are "major":

```
!-----
! initializations
!-----
```

while those indented to match the following pertinent code are "minor":

```
    ! compute aggregate ice state and open water area
    call aggregate
```

- Short comments can be included on the same line as the source code preceded by the "!" character.
- Comments on adjacent lines should be aligned, if possible.

## Portability

- The code will run on all platforms supported by CCSM. See the CCSM User's Guide for the most recent list of supported machines and platforms.
- Code should be developed using standard MPI (Message Passing Interface).
- The code should run efficiently on cache-based and vector machines.
- Standard F90 and MPI should be used so that the code is easily portable to other platforms.

## Incomplete and dead code

- Incomplete code should be indicated by comments that include "TBD".
- Variables that are declared but not used should be removed.
- Code that is commented out should either be removed or contain comments as to why it is still there.

## Miscellaneous

- Tabs should not be used for spacing; they create a large inconvenience when editing.
- Code will be written in lower case. This differentiates Fortran code from C preprocessor tokens, since the convention has been established that these are upper case.
- Use of the relational operators `<`, `>`, `<=`, `>=`, `==`, and `/=` are preferred to `.lt.`, `.gt.`, `.le.`, `.eq.`, and `.ne.` for purposes of readability.
- The `stop` command should be avoided when exiting the code after an error. Instead, the subroutine `abort_ice` should be called.

## 5.2 Content Guidelines

- **Sign Conventions** Enthalpy is defined as negative; fluxes are positive downward.
- **Implicit None** All modules will include an `implicit none` statement, immediately following the `use` statements. This implies that all variables must be explicitly typed.
- **Prologues** Each module, subroutine and function will include a prologue for use with the ProTeX auto-documentation script (<http://dao.gsfc.nasa.gov/software/protex>). This will be used to make a code reference document.
- **I/O Error Conditions** I/O statements which need to check an error condition, i.e. namelist read statements, should use the `iostat = <integer variable>` construct instead of `end` and `err`.
- **Intent** All dummy arguments should include the `INTENT` clause in their declaration.
- **Conditionals** Conditionals based on `< 0.` or `> 0.` should be avoided.
- **Physical Constants** Values of physical constants should not be hardwired into the executable portion of a code. Constants should be defined in a single module as named, full-precision parameters.
- All units should be in MKS; CGS conversions should be avoided. Diagnostics and other fields that do not affect the model integration may be in other units.

## 6 Software Integration Procedure

This section outlines the steps for testing and integrating newly developed code into the existing sea ice model. This procedure has been designed to ensure that new code meets CCSM and CSIM requirements and improves the climate simulation and/or the model performance. All substantial code changes must go through the Polar Climate Working Group (**PCWG**) review process. Substantial refers to all physics changes and any software engineering changes that result in major changes to code organization and/or efficiency. This generally does not include cosmetic changes to the code.

It may not be possible or necessary for a developer to carry out all of the steps described in this section. For most development, it will be necessary to contact the liaison or a co-chair of the Polar Climate Working Group. This information is on the PCWG web page:

[http://www.cesm.ucar.edu/working\\_groups/Polar/](http://www.cesm.ucar.edu/working_groups/Polar/).

The PCWG review process will usually involve the following steps:

1. Modeler develops improved code. This can include modification of a few lines, improving a parameterization, or providing a new physics module.
2. Modeler tests changes within CCSM framework. See sections 6.1 and 6.2.
3. Outcome of tests is posted on PCWG web site and modeler provides explanation to PCWG members.

4. PCWG members review results (and code if desired).
5. PCWG decide whether to recommend adoption of change to CCSM Code Review Board.

## 6.1 Making a Control Run

Before new code is put into CSIM, a control run using the latest tagged version of CCSM may need to be done to document the effects of the new code on model performance and the simulated climate. If newly developed code is being put into a tagged version of CCSM, output from a standard control run may already be available, in which case, this set of steps can be skipped. If output from a control run is not available, here are the steps to create it:

1. Get a tagged version of CCSM via one of the following options:
  - (a) At **NCAR**:
    - i. Check it out from CVS: `cvs co -r ccsm_tag_name`
    - ii. Copy it from `/fs/cgd/csm/collections`
  - (b) Off site:
    - i. Download it from <http://www.cesm.ucar.edu/models/ccsm3>
    - ii. Contact the PCWG liaison or a co-chair for a copy.

If you have questions regarding the appropriate CCSM tag name to use, contact a PCWG co-chair or the liaison.

2. See the CCSM User's Guide on how to set up the scripts for the ***M component set*** (latm, dlnd, docn, csim (mixed layer ocean mode), cpl).
3. Modify the latm source code and setup script to use your favorite forcing. The PCWG can provide reasonable input datasets if needed.
4. Do a 10-15 year simulation to get a control run case, saving output in a permanent disk location.
5. Document performance by saving the timing information at the bottom of a log file, and put plots of output on web if necessary. A graphics package is available for plotting output.

## 6.2 Testing Climate and Performance of Modified Code

For purposes of adding new code to the existing model, a tagged version of CCSM should be used, the more recent the better. The latest version of CCSM will usually contain the latest version of CSIM. It is a good idea to check if this is the case. These are the steps for integrating newly developed code into the existing sea ice model and testing:

1. Make modifications to the source code using one of the following options:
  - (a) Using CVS:
    - i. Create a development branch from a CSIM tag on the main trunk, ideally the CSIM tag used in the control run.
  - (b) Not using CVS:
    - i. Copy modules to be modified from the source code directory to `ccsm3/scripts/$CASE/SourceMods/src.csim`, and make modifications in this directory.
2. Compile, run for 5 days with minimum debugging turned on: array bounds check, Nan initialization, floating point trap.
3. Check that code still meets any requirements that may have been affected by the new code. The minimum tests are exact restart and conservation. If developed code involves any namelist variables, all options for those variables must be tested. Any tests available in the new code should be exercised.

4. Test on all supported platforms. Compilation and a 5 day run are the minimum.
5. Do a 1 year run, check results against control run.
6. Continue run out to 10-15 years, saving output in a permanent disk location. This may be used as the next control run.
7. Document performance, compare with control run and put difference plots on web.
8. If results are satisfactory (approved by PCWG), code may be inspected and/or reviewed. Development branch will be merged with the main trunk and tagged.
9. Model documentation (User's Guide, Code Reference and Scientific Description) must be updated.

Currently, there is no formal CCSM Requirements document, and the CSIM Requirements document is still in the draft stage. There is a somewhat complete list of tests for CSIM, but an automated test suite is still under development. All developers are strongly encouraged to read the CSIM Requirements Document and all of the Developer's Guide (this document).

### Minimum Requirements

It is difficult to define all tests that are suitable for all code changes, and a user may not have access to all supported platforms. This section gives a list of basic requirements for the ice model, to provide guidance for testing. A more complete list of requirements is in the CSIM Requirements document (under development). As noted in section 6.2, any changes in climate or model performance due to the code modifications must be documented.

**Minimum CCSM Requirements** Compiles, runs, and restarts exactly with at least one configuration, preferably the *M component set* (latm, dlnd, docn, csim in mixed layer ocean mode, cpl) or the *B component set* (cam, clm, pop, csim, cpl). Runs on at least one standard CCSM grid (gx1v3 or gx3v5). Runs on all CCSM supported platforms available to the modeler.

**Conservation** checking water, salt and energy budgets in the log file. Any conservation errors or CFL violations that occurred in the 10-15 year climate year run should be documented.

**Exact Restart** This test is automated in the CSIM and CCSM scripts. It does a 10 day startup run and a 5 day continuation run that begins at day 5 of the startup run. Output from days 5-10 of both runs are compared and must match bit-for-bit.

**Debug Test** This test is also automated in the CSIM and CCSM scripts.

**Input Data Files** If any new or modified input data files are necessary to run the modified code, they must be provided with the code.

## 7 Glossary

**B component set** The shorthand name for the combination of all active components of CCSM: ocean, atmosphere, land, and ice exchanging information via a coupler.

**CCSM** The Community Climate System Model is a fully-coupled, global climate model that provides state-of-the-art computer simulations of the Earth's past, present, and future climate states.

**CICE** The Los Alamos sea ice model

**CSIM** The CCSM Community Sea Ice Model

**CSIM4** Version 4 of the CCSM Community Sea Ice Model that was released in May 2002 with CCSM2.0.

**CSIM5** Version 5 of the CCSM Community Sea Ice Model that was released in June 2004 with CCSM3.0.

**CVS** The Concurrent Versions System used to record the history of source code.

**CVS branch** A line of development separate from the main trunk.

**CVS main trunk** The main line of development on the CVS source tree.

**inline** An optimization where the code of called routines is inserted into the calling code to eliminate the calling overhead.

**LANL** Los Alamos National Laboratory

**load balancing** The distribution of processing and communications activity between components to minimize the resources used and the time spent waiting by any single component.

**M component set** The shorthand name for the combination of data ocean, atmosphere, and land models, csim with the mixed layer ocean, exchanging information via a coupler.

**MPI** Message Passing Interface is the protocol for passing messages between parallel processors.

**NCAR** National Center for Atmospheric Research, in Boulder, Colorado

**PCWG** The Polar Climate Working Group is a team of scientists who develop and improve the sea ice component of CCSM, and who use the ice model alone or fully coupled in CCSM for studies of polar climate.

**ProTeX** A perl script that allows for automatic generation of Latex compatible documentation of source code, without a considerable effort beyond the documentation of the code itself.

## References

Hunke, E. C. and W. H. Lipscomb, 2004: *CICE: the Los Alamos Sea Ice Model, Documentation and Software User's Manual*. T-3 Fluid Dynamics Group, Los Alamos National Laboratory, Tech. Rep. LA-CC-98-16 v.3.1.

## A Calling Tree

This section contains the primary calling tree for the coupled ice model. Calls to MPI, netCDF, share code routines, subroutines or functions within the ice model that do not include ice physics (i.e. global\_scatter, get\_sum, bound, etc.) are not included. Calls to the slab ocean mixed layer model within the ice model are shown in square brackets.

```
ICEMODEL
|
|--SETUP_MPI--ICE_COUPLING_SETUP
|--SHR_MSG_CHDIR
|--SHR_MSG_DIRIO
|--INIT_CONSTANTS
|--INPUT_DATA
|--INIT_GRID
|  |--POPGRID
|  |--COLUMNGRID
|  |--RECTGRID
|  |--TLATLON
|  |--MAKEMASK
|--INIT_REMAP
|--INIT_CALENDAR
|--INIT_HIST
|--INIT_EVP
|--INIT_FLUX--INIT_FLUX_ATM
|  |--INIT_FLUX_OCN
|--INIT_THERMO_VERTICAL
|--INIT_ITD
|--INIT_MECHRED--COMP_MATRICES
|--INIT_STATE--AGGREGATE
|  |--BOUND_AGGREGATE
```

```

+-RESTARTFILE+-AGGREGATE
|
|   +-BOUND_AGGREGATE
+-ALBEDOS
+-CALENDAR
+-[INIT_OCEANMIXED_ICE]
+-INIT_CPL
+-INIT_DIAGS
+-ICE_WRITE_HIST+-PRINCIPAL_STRESS
|
|   +-ICECDF - write IC to netCDF file
|-----|
| BEGIN TIMESTEP LOOP
|-----|
+-FROM_COUPLER
+-[SET_OCEANMIXED_ICE]
+-INIT_MASS_DIAGS
+-THERMO_VERTICAL
| +-INIT_FLUX_ATM
| +-INIT_DIAGNOSTICS
| +-FRZMLT_BOTTOM_LATERAL
|-----|
| Loop through categories
|-----|
| +-INIT_COUPLING_VARS
| +-ATMO_BOUNDARY_LAYER
| +-INIT_VERTICAL_PROFILE
| +-ADD_NEW_SNOW
| +-TEMPERATURE_CHANGES+-CONDUCTIVITY
| |
| |   +-ABSORBED_SOLAR
| |   +-SURFACE_FLUXES
| |   +-TRIDIAG_SOLVER
| +-THICKNESS_CHANGES
| +-MERGE_FLUXES
| +-UPDATE_STATE_VTHERMO
|-----|
| End loop through categories
|-----|
+-SCALE_FLUXES
+-TO_COUPLER
+-[TIME_INTRPLT_OCEAN_FORCING+-OCNHEAT+-ATMO_BOUNDARY_LAYER]
+-THERMO_ITD+-AGGREGATE
|
|   +-BOUND_AGGREGATE
|   +-INIT_FLUX_OCN
|   +-REDUCE_AREA
|   +-REBIN+-SHIFT_ICE
|   +-ZAP_SMALL_AREAS+-NORMALIZE_STATE+-AGGREGATE
|   +-LINEAR_ITD+-AGGREGATE_AREA
|   |
|   |   +-COLUMN_SUM
|   |   +-COLUMN_CONSERVATION_CHECK
|   |   +-SHIFT_ICE
|   |   +-FIT_LINE
|   +-ADD_NEW_ICE+-COLUMN_SUM
|   |
|   |   +-COLUMN_CONSERVATION_CHECK
|   +-LATERAL_MELT
|   +-FREEBOARD
+-EVP+-EVP_PREP
|
|   +-STRESS
|   +-STEPU
|   +-EVP_FINISH
+-TRANSPORT_REMAP+-DEPARTURE_POINTS
|
|   +-LOCATE_TRIANGLES
|   +-TRIANGLE_COORDINATES
|   +-MAKE_MASKS
|   +-CONSERVED_SUMS
|   +-CONSTRUCT_FIELDS--LIMITED_GRADIENT
|   +-FLUX_INTEGRALS
|   +-UPDATE_FIELDS
|   +-GLOBAL_CONSERVATION
|   +-LOAD_TRACERS
|   +-LOCAL_MAX_MIN
|   +-CHECK_MONOTONICITY
|   +-UNLOAD_TRACERS
+-TRANSPORT_MPDATA+-CHECK_STATE
|
|   +-MPDATA
+-RIDGE+-RIDGE_MATRICES--COMP_MATRICES
|
|   +-RIDGING_MODE
+-ZAP_SMALL_AREAS
+-REBIN
+-AGGREGATE
+-BOUND_AGGREGATE
+-ALBEDOS
+-SCALE_HIST_FLUXES

```

```
+--RUNTIME_DIAGS
+--ICE_WRITE_HIST--PRINCIPAL_STRESS
|
|      +-ICECDF
+-DUMPFILE
|-----
| END TIMESTEP LOOP
|-----
+-EXIT_COUPLER
```

## B ProTeX Generated Documentation

This appendix contains the module and subroutine documentation generated by ProTeX. This section is available at

[http://www.cesm.ucar.edu/models/ccsm3/csim/RefGuide/ice\\_refdoc/index.html](http://www.cesm.ucar.edu/models/ccsm3/csim/RefGuide/ice_refdoc/index.html) .