CSIM Code Reference Manual
Version 4

by Bruce P. Briegleb[1]

May 22, 2002

[1]

[1]National Center for Atmospheric Research, PO Box 3000, Boulder CO 80307 (NCAR is sponsored by the National Science Foundation)

Contents

1 Introduction

The Community Climate System Model (CCSM) is a coupled climate model consisting of atmosphere, ocean, land, and sea ice components which exchange fluxes and states through a coupler. This model is making important contributions to knowledge about past, present, and future climates.

The sea ice component of CCSM Version 2 is the Community Sea Ice Model Version 4, or CSIM4. The physics of this component is described in Briegleb et al. (2002). Accessing and running CSIM4 is described in Schramm (2002). This document discusses the CSIM4 source code, its general organization and data structures, a detailed listing of all modules (including subroutines, module reference and input/output arguments), how to add new fields to the history file, and finally suggestions on modifying the source code.

2 Overview of CSIM4 Code

The CSIM4 source code is maintained in the Concurrent Version System (CVS) at NCAR. It can be accessed off the web, or by a CVS checkout (co) command online at NCAR. The source code is written in Fortran 90, and consists of thirty-one files all resident in a single source directory. The main driver file is 'ice.F', which runs the sea ice model.

Each source code file contains a single module of the same name as the file (excluding the suffix .F), with the exception of the main driver file (*ice.F*). For example, the file *ice_scaling.F* contains the module *ice_scaling*. Within most modules are various data declarations and subroutines, along with protex commands for automatic documentation.

The modules can be organized as in Table 1 in the approximate order of call in the driver file *ice.F*. The four modules at the end of the table refer to optional and io routines. The logical structure of the code is: initialization and time stepping loop, which involves receiving forcing data from the coupler, a partial thermodynamic calculation to compute the flux exchange with the atmosphere, sending of the output to the coupler, completion of the thermodynamic calculation, linear remapping, calculation of the ice velocities, advection, mechanical redistribution and albedos. Note that while this logical structure is present in the driver *ice.F*, the reader must wade through the options of prescribed ice and ocean mixed layer to actually recognize it as these options alter the calling sequence.

This particular order of computations is necessary for coupled model accuracy as well as load balancing. For accuracy in radiant energy absorbed, the albedos used by the atmosphere in calculating the down shortwave should be the same as those used in the surface energy absorption in the sea ice thermodynamics. This requires that the snow/ice albedos be updated in the sea ice model *after* the thermodynamic calculation with the down shortwave is completed. For load balancing, the atmosphere and sea ice can run in parallel more efficiently if the atmosphere-ice surface exchange is separated from the rest of the sea ice thermodynamics. Hence, the sea ice model computes the atmosphere-ice fluxes as soon as it receives forcing data from the coupler, and returns the output fluxes to the coupler as soon as they are computed.

Because the sea ice model domain is limited globally, further improvements in load balancing arise from limiting the total amount of data exchanged between the sea ice model and the coupler (further explained in the Data Structure's section).

It should be noted that the coupler requires the fluxes to be in the form of per unit ice area (for it summation over ocean/land/ice fluxes in exchange with the atmosphere). Thus, the atmosphere-ice fluxes computed in the ice model (aggregated over thickness distribution) are further divided by the aggregate ice area (see module *ice_scaling*) before being sent to the coupler.

Table 1: Source Code Modules

Module	Description
ice.F	Main driver
—ice_mpi_internal.F	Sets MPI tasks
—ice_timers.F	Timers for performance
—ice_model_size.F	Defines global domain, category, layer sizes
—ice_kinds_mod.F	Definitions of common data types
—ice_state.F	Defines ice state variables
—ice_domain.F	Local domain sizes; parallel info
——ice_constants.F	Sets physical constants
——ice_init.F	Parameter and variable initilizations
——ice_grid.F	Spatial grids, masks and boundary conditions
——ice_calendar.F	Calendar routines for managing time
——ice_flux.F	Assigns flux arrays: from, to coupler and internal
——ice_itd.F	Initilizes and redistributies ice in the itd
——ice_therm_driver.F	Driver for thermodynamics and itd
———ice_atmo.F	Calculate atmosphere-ice stability based fluxes
———ice_vthermo.F	Sets up vertical thermodynamics
———ice_tstm.F	Energy conserving thermodynamics
———ice_dh.F	Grow/melt snow/ice and adjust temperature profile
———ice_itd_linear.F	Linear remapping scheme for itd
——ice_scaling.F	Scale fluxes to coupler
——ice_coupling.F	Message passing to and from coupler
——ice_dyn_evp.F	EVP dynamics
——ice_transport.F	Advection
——ice_mechred.F	Mechanical redistribution
——ice_albedo.F	Snow/Ice albedo
——ice_history.F	Restart and history files
——ice_diagnostics.F	Diagnostic printout
—ice_prnpnt.F	Print data for selected points
—ice_read_write.F	Read/write io
—ice_ocean.F	Slab ocean mixed layer
—ice_prescribed.F	Prescribed ice

2.1 Calling Tree

This section contains the primary calling tree for the coupled ice model. Calls to MPI, netCDF, and share code routines are not included as well as calls to subroutines or functions within the ice model that do not include ice physics (i.e. global_scatter, get_sum, bound, etc.). Calls to the prescribed ice routines and the ocean mixed layer model within the ice model are shown in parentheses.

```
ICEMODEL
+-SETUP_MPI-+-MPI_COUPLED
-----------------------
Initialize model
-----------------------
+-INIT_CONSTANTS
+-INPUT_DATA
+-INIT_GRID
| +-POPGRID-+-ICE_OPEN
| |         +-ICE_READ
| +-TLATLON
| +-MAKEMASK
+-INIT_CALENDAR
+-INIT_HIST
+-INIT_EVP
+-INIT_FLUX-+-INIT_FLUX_ATM
|           +-INIT_FLUX_OCN
+-INIT_ITD
+-INIT_THERMO
+-INIT_MECHRED-+-COMP_MATRICES
+-INIT_STATE-+-AGGREGATE
+-(INIT_PRESCRIBED_ICE)
+-RESTARTFILE-+-AGGREGATE
+-ALBEDOS
+-CALENDAR
+-(INIT_OCEANMIXED_ICE)
+-INIT_CPL-+-CALENDAR
|          +-TO_COUPLER-+-DIAGS_CPL_OUT
+-INIT_DIAGS
+-INTPNT
+-ICE_WRITE_HIST+-ICECDF
-----------------------
Begin timestepping loop
-----------------------
+-FROM_COUPLER-+-T2UGRID
|              +-DIAGS_CPL_IN
+-(SET_OCEANMIXED_ICE)
+-(READ_PRESCRIBED_ICE_CLIMATE)
+-(READ_PRESCRIBED_ICE)
+-(SET_PRESCRIBED_ICE_STATE)-+-(AGGREGATE)
+-(SET_PRESCRIBED_ICE_FLUX)-+-(INIT_FLUX_ATM)
|                           +-(INIT_FLUX_OCN)
+-INIT_MASS_DIAGS
+-THERMO_RATES-+-INIT_COLUMN_DIAGS
|              +-TO_COLUMN
|              +-INIT_FRZMLT
|              +-INIT_VERTICAL_PROFILE
```

```
|              +-THERMO_VERTICAL-+-STABILITY
|              |                 +-ICE_SFC_FLUX
|              |                 +-TSTM
|              |                 +-DH
|              |                 +-MERGE_FLUXES
|              +-FROM_COLUMN
+-SCALE_FLUXES
+-TO_COUPLER-+-DIAGS_CPL_OUT
+-(TIME_INTRPLT_OCEAN_FORCING)
+-(COMPUTE_OCEANMIXED_ICE)-+-(MIXED_LAYER)-+-(STABILITY)
|                                           +-(OCNHEAT)
+-THERMO_ITD-+-INIT_FLUX_OCN
|            +-TO_COLUMN
|            +-LINEAR_ITD-+-AGGREGATE_PT
|            |            +-PRINT_STATE-+-TO_COLUMN
|            |            +-FIT_LINE
|            |            +-SHIFT_ICE
|            +-LATERAL_GROWTH_MELT
|            +-REDUCE_AREA
|            +-REBIN_ICE-+-MOVEUP--ZEROCAT
|            |           +-MOVEDN--ZEROCAT
|            +-AGGREGATE_PT
|            +-CONSERVATION_CHECK-+-PRINT_STATE
|            +-FROM_COLUMN
|            +-CHECK_STATE
+-EVP-+-BOUND_SW--BOUND_IJN
|     +-EVP_PREP-+-T2UGRID-+-BOUND
|     |          |         +-TO_UGRID
|     |          +-TO_UGRID
|     +-STRESS
|     +-STEPU-+-BOUND_NARR_NE--BOUND_NIJ
|     +-EVP_FINISH-+-U2TGRID-+-BOUND
|                            +-TO_TGRID
+-TRANSPORT-+-BOUND_NARR--BOUND_IJN
|           +-MPDATA-+-BOUND_NARR
+-MECHANICAL_REDISTR-+-TO_COLUMN
|                    +-RIDGE-+-RIDGE_MATRICES-+-COMP_MATRICES
|                    |       +-RIDGING_MODE
|                    +-ROTH_STRENGTH-+-RIDGING_MODE
|                    +-FROM_COLUMN
+-DISTR_CHECK-+-TO_COLUMN
|             +-MOVEUP
|             +-MOVEDN
|             +-REBIN_ICE
|             +-FROM_COLUMN
|             +-NORMALIZE_STATE
+-AGGREGATE
+-SCALE_HIST_FLUXES
+-RUNTIME_DIAGS
+-DUMPFILE-+-ICE_OPEN
|          +-ICE_WRITE
-----------------------
End timestepping loop
-----------------------
```

```
+-(EXIT_PRESCRIBED_ICE)
+-EXIT_COUPLER
```

3 Data Structures

Both horizontal and vertical discretization of the sea ice model is presented here, as well as category discretization. Two forms of horizontal discretization are distinguished: a global 2D domain decomposition, and an exclusion of lower latitude regions of no ice in the exchange with the coupler. The ice thickness category discretization can be thought of as a horizontal discretization by ice thickness. The vertical discretization varies with ice thickness category.

3.1 Horizontal

The horizontal computational grid is decomposed into two dimensions for efficiency in parallelization. The global horizontal domain of dimensions $imt_{global} \times jmt_{global}$ (for example, the gx1 grid has $imt_{global} = 320$ longitude points and $jmt_{global} = 384$ latitude points) is divided into integral NX longitude by NY latitude rectangular subdomains of dimensions $(imt_{local} = imt_{global}/NX + 2n_{ghost} + 1) \times (jmt_{local} = jmt_{global}/NY + 2n_{ghost} + 1)$, where imt_{global}/NX, jmt_{global}/NY must be integers (see module *ice_domain*). Each subdomain has a physical portion indexed as $[ilo : ihi, jlo : jhi]$ with n_{ghost} boundary cells outside. Periodic boundary conditions are applied, with boundary routines performing communications between subdomains when running parallel. Global scatter and gather routines distribute information from the global domain to the subdomains and back, respectively (see the "bound" routines in module *ice_grid*).

The thermodynamic calculations involve one grid point at a time (see module *ice_therm_driver*) so that a purely thermodynamic model integration is bit-for-bit independent of the domain decompostion (i.e. the exact values of NX, NY), while the dynamic calculation depends bit-for-bit upon domain boundary conditions, and therefore on the exact values of NX and NY. The sea ice model climate is not sensitive to the precise domain decomposition.

The actual source code uses $NPROC\_X$ and $NPROC\_Y$ for NX and NY respectively, referring to the number of processors in the x direction and in the y direction (these parameters are set by a pre-compiler; see the CSIM User's Guide Version 4). The module *ice_domain* defines the local subdomain dimensions and their beginning/ending indices. Module *ice_init* computes and assigns the subdomain processor numbers. The module *ice_model_size* assigns the global domain values.

The module *ice_coupling* handles the 'unpacking' and 'packing' of the data horizontally after and before exchange with the coupler. This involves excluding latitude bands around the equator that have no sea ice present. Information on the number of latitude bands of data for each hemisphere (i.e. the number of latitude bands of actual hemispheric sea ice but with sufficient equatorwards bands to allow for ice formation along ice edge as well) is included in the data exchange header. As the areas where new ice formation will occur are known to the coupler (based on the freeze/melt potential received from the ocean model), the coupler is able to determine the precise latitude limits. Regions with no ice in the unpacked fields are simply filled with zeros after being received from the coupler.

3.2 Category

The number of ice thickness categories (*ncat*) is assigned in module *ice_model_size*. The state variables which vary with category are found in module *ice_state*. Note that only aggregate quantities (i.e. averaged over the ice thickness distribution) are exchanged with the coupler.

3.3 Vertical

Vertically, sea ice is discretized into L evenly spaced layers, where the number of layers of ice (L) depends on the thickness category, with the thinnest two categories having two equally spaced layers and the thicker three categories having four. The total sum of vertical levels for all categories is specified in module *ice_model_size* as *ntilay*, with the maximum number of levels for any category being *nmax*. Several state variables are dependent on vertical level, as seen in module *ice_state*.

4 Module Descriptions

In this section, we present a detailed module-by-module listing of the CSIM4 source code. This text was generated using protex, which uses embedded key words in the source code itself. Using the protex commands and structure allows one to maintain and update internal documentation in the source code from which external code documentation can be generated. See Table 2 for the protex keywords embedded in the source code. Each protex keyword and description text is headed by exclamation mark (!), and ends with a colon (:), excepting the beginning and ending keywords. All protex keywords and description lines must lie within the protex beginning and ending keywords. Within a description section, latex is used to interpret the text, and so in particular two backslashes are used as carriage return, and backslash underscore is used for underscore.

Table 2: Protex Keywords

Keyword	Description
!BOP	Beginning of protex section
!EOP	End of protex section
!MODULE:	Module name follows on same line
!DESCRIPTION:	Description section in latex
!REVISION HISTORY:	Gives author and revision information
!INTERFACE:	Actual fortran module or subroutine statement
!USES:	Sets off module uses declaration section
!IROUTINE:	Routine name and one line description
!INPUT/OUTPUT PARAMETERS:	Input/Output declaration section follows

4.1 Fortran: Module Interface icemodel - main ice model program (Source File: ice.F)

The NCAR Community Climate System Model Version 2 - Sea Ice (CCSM2-SI) is intended for use by the climate modeling community, and its development has been a true community effort. The list of physics to be included in CCSM2-SI was developed by the CCSM Polar Climate Working Group (PCWG) in a series of meetings during 1997-1999. The specific parameterizations and codes to be implemented were identified and agreed upon at a meeting of ice model developers from the PCWG, held at NCAR in September, 1999.

A brief description of the development of the CCSM2-SI code is as follows:

o The code originated from the LANL CICE model (version of September, 1999). CCSM2-SI retains the following elements from this original code: Elastic-viscous-plastic dynamics (EVP), MPDATA transport scheme, and much of the original code structure.

o Later in autumn of 1999, the code was changed as follows: (1) the MPI parallelization routines were updated; (2) a multi-category ice thickness distribution was added; (3) a new thermodynamic code with explicit treatment of brine pockets was added.

o In late 1999/early 2000, changes were made to the history output.

o During winter-spring 2000, the Active Ice Only (AIO) framework was developed, together with datasets to be used as forcing within this framework. The model was tested within this AIO framework.

o In the fall of 2000, the code was streamlined, checked, standardized, and updated to Fortran 90. Documentation is currently being written.

The development team for the model described above is:

Elizabeth Hunke, LANL
Cecilia Bitz, U. WA.
Bruce Briegleb, NCAR
Tony Craig, NCAR
Marika Holland, NCAR
Bill Lipscomb, LANL
Julie Schramm, NCAR

Numerous others have contributed to this effort also–thanks to all!

INTERFACE:

```
program icemodel
```

USES:

```
use ice_timers
use ice_mpi_internal
use ice_albedo
use ice_vthermo
use ice_mechred
use ice_scaling
use ice_diagnostics
use ice_history
use ice_coupling
use ice_grid
use ice_calendar
use ice_dyn_evp
use ice_itd
use ice_itd_linear
use ice_transport
use ice_therm_driver
use ice_init
use shr_msg_mod,only:shr_msg_chdir,shr_msg_dirio,shr_sys_flush
use ice_prescribed
use ice_ocean
use ice_prnpnt
```

REVISION HISTORY:

```
February 2001 - E.Hunke (LANL) provided original
May 2001      - Final form settled (M.Holland, J.Schramm and
                B.Briegleb, NCAR)
```

4.2 Fortran: Module Interface ice_albedo - snow and ice albedo parameterization (Source File: ice_albedo.F)

The albedo parameterization

REVISION HISTORY:

```
authors:  Bruce P. Briegleb, NCAR
          Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_albedo
```

USES:

```
use ice_kinds_mod
use ice_domain
```

4.2.1 albedos - compute snow/ice albedos and aggregate

INTERFACE:

```
subroutine albedos
```

DESCRIPTION:

Compute albedos and aggregate them
note: ice albedo is zero if no ice present

REVISION HISTORY:

```
authors:  Bruce P. Briegleb, NCAR
          Elizabeth C. Hunke, LANL
```

USES:

```
use ice_constants
use ice_grid
use ice_state
```

INPUT/OUTPUT PARAMETERS:

4.3 Fortran: Module Interface ice_atmo - atm-ice interface: stability based flux calculations (Source File: ice_atmo.F)

Atmospheric boundary interface (stability based flux calculations)

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_atmo
```

USES:

```
use ice_domain
use ice_constants
use ice_flux
use ice_state
```

4.3.1 stability - compute coefficients for atm-ice fluxes, stress and Tref

INTERFACE:

```
      subroutine stability( i,j,nc, Tsf, rdn_in,
     $         strx   ,stry   ,Trf, dssqdt,  delt, delq)
```

DESCRIPTION:

Compute coefficients for atm/ice fluxes, stress, and reference temperature. NOTE:
(1) all fluxes are positive downward,
(2) net heat flux = fswabs + flwup + (1-emissivity)flwdn + fsh + flh,
(3) here, tstar = (WT)/U*, and qstar = (WQ)/U*,
(4) wind speeds should all be above a minimum speed (eg. 1.0 m/s).

ASSUME:
The saturation humidity of air at T(K): qsat(T) (kg/m**3)

Code originally based on CSM1

REVISION HISTORY:

```
   author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: i,j,nc

      real (kind=dbl_kind), intent(in) ::
     &   Tsf    ! surface temperature of ice or ocean
     &,  rdn_in ! initial value for rdn, dependent on ice/ocean surface

      real (kind=dbl_kind), intent(out) ::
     &   strx   ! x surface stress (N)
     &,  stry   ! y surface stress (N)
     &,  Trf    ! reference height temperature  (K)
     &,  dssqdt ! derivative of ssq wrt Ti (kg/kg/K)
     &,  delt   ! potential T difference   (K)
     &,  delq   ! humidity difference      (kg/kg)
```

4.3.2 ice_src_flux - compute ice-atm surface fluxes

INTERFACE:

```
      subroutine ice_sfc_flux(i,j,nc, dssqdt, delt, delq,
     &    flwdabs,flwup, fswabs,fswabsv,fswabsi,
     &    fsh,flh,dflhdT,dfshdT,dflwdT)
```

DESCRIPTION:

Compute ice-atm surface fluxes

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_kinds_mod
use ice_albedo
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i,j,nc

real (kind=dbl_kind), intent(in) ::
&     dssqdt ! derivative of ssq wrt Ti (kg/kg/K)
&,    delt   ! potential T difference   (K)
&,    delq   ! humidity difference      (kg/kg)

real (kind=dbl_kind), intent(out) ::
&     flwdabs   ! down long-wave  absorbed heat flx   (W/m**2)
&,    flwup     ! emitted long-wave upward heat flux  (W/m**2)
&,    fswabs     ! srfc absrbd short-wave heat flux (W/m**2)
&,    fswabsv    ! fswabs in vis (wvlngth < 700nm)  (W/m**2)
&,    fswabsi    ! fswabs in nir (wvlngth > 700nm)  (W/m**2)
&,    fsh      ! sensible         heat flux  (W/m**2)
&,    flh      ! latent           heat flux  (W/m**2)
&,    dflhdT     ! d(flh)/d(T)      (W/m**2/K)
&,    dfshdT     ! d(fsh)/d(T)      (W/m**2/K)
&,    dflwdT    ! d(flwup)/d(T)     (W/m**2/K)
```

4.4 Fortran: Module Interface ice_calendar - calendar routines for managing time (Source File: ice_calendar.F)

Calendar routines for managing time

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         Tony Craig, NCAR
```

INTERFACE:

```
module ice_calendar
```

USES:

```
use ice_constants
```

4.4.1 init_calendar - initialize calendar variables

INTERFACE:

```
subroutine init_calendar
```

DESCRIPTION:

Initialize calendar variables

REVISION HISTORY:

```
   authors: Elizabeth C. Hunke, LANL
            Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

4.4.2 calendar - computes date at the end of the time step

INTERFACE:

```
      subroutine calendar(ttime)
```

DESCRIPTION:

Determine the date at the end of the time step

REVISION HISTORY:

```
   authors: Elizabeth C. Hunke, LANL
            Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   ttime                                ! time variable
```

4.5 Fortran: Module Interface ice_constants - sets physical constants (Source File: ice_constants.F)

This module defines a variety of physical and numerical constants used throughout the ice model

Code originally based on constants.F in POP

REVISION HISTORY:

```
   author Elizabeth C. Hunke, LANL
```

INTERFACE:

```
      module ice_constants
```

USES:

```
#ifdef coupled
      use shr_const_mod
#endif
      use ice_kinds_mod
      use ice_domain
```

4.5.1 init_constants - initialize constants defined at run time

INTERFACE:

```
subroutine init_constants
```

DESCRIPTION:

Initializes constants that are best defined at run time (e.g. pi)

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

4.6 Fortran: Module Interface ice_coupling - message passing to and from the coupler (Source File: ice_coupling.F)

Message passing to and from the coupler

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_coupling
```

USES:

```
use ice_calendar
use ice_model_size
use ice_prescribed
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

4.6.1 init_cpl - initializes message passing between ice and coupler

INTERFACE:

```
subroutine init_cpl
```

DESCRIPTION:

Initializes message passing between ice and coupler

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_domain
use ice_constants
use ice_mpi_internal
use ice_grid
use ice_calendar
use ice_history, only : runtype
```

INPUT/OUTPUT PARAMETERS:

4.6.2 from_coupler - input from coupler to sea ice model

INTERFACE:

```
subroutine from_coupler
```

DESCRIPTION:

Reads input data from coupler to sea ice model

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_domain
use ice_constants
use ice_flux
use ice_timers
use ice_mpi_internal
use ice_diagnostics
use ice_grid
use ice_calendar
use ice_state
use ice_prnpnt
```

INPUT/OUTPUT PARAMETERS:

4.6.3 to_coupler - send data from sea ice model to coupler

INTERFACE:

```
subroutine to_coupler
```

DESCRIPTION:

Sea ice model to coupler

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_model_size
use ice_domain
use ice_constants
use ice_flux
use ice_timers
use ice_mpi_internal
use ice_albedo
use ice_diagnostics
use ice_grid
use ice_calendar
use ice_state
use ice_history, only : sabs
use ice_prnpnt
use ice_prescribed
use ice_ocean
```

INPUT/OUTPUT PARAMETERS:

4.6.4 exit_coupler - exit from coupled/mpi environment

INTERFACE:

```
subroutine exit_coupler
```

DESCRIPTION:

Exit from coupled/MPI environment

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_kinds_mod
use ice_model_size
use ice_domain
```

INPUT/OUTPUT PARAMETERS:

4.6.5 mpi_coupled - sets mpi communicators and component task ids

INTERFACE:

```
subroutine mpi_coupled (in_model_name,
&                        cpl_task, model_task, model_comm)
```

DESCRIPTION:

This routine queries all the components of the full coupled
system and sets up proper communicators and task ids for each
component of the coupled system

This routine should be called after mpi-init, but before
setting up any internal mpi setups (since these will require
the internal communicators returned by this routine)

Code originally based on POP routine

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
      include "mpif.h"                  ! MPI library definitions
```

INPUT/OUTPUT PARAMETERS:

```
      character (3), intent(in) :: in_model_name
                       ! 3-letter identifier (atm, lnd, ocn, ice, cpl)
                       ! for the model calling this routine

      integer, intent(out) ::
     &  cpl_task           ! master task of coupler
     &, model_task         ! master task of model (in MPI_COMM_WORLD)
     &, model_comm         ! communicator for internal model comms
```

4.7 Fortran: Module Interface ice_dh (Source File: ice_dh.F)

Energy-conserving sea ice model
Routines to grow/melt ice and adjust temperature profile

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669-15,677.

See Bitz, C.M., M.M. Holland, M. Eby, and A.J. Weaver, 2001: Simulating the ice-thickness distribution in a coupled climate model, J. Geophys. Res., 106, 2441-2464.

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

INTERFACE:

```
      module ice_dh
```

USES:

```
      use ice_kinds_mod
      use ice_constants
      use ice_itd
```

4.7.1 dh - computes top and bottom thickness changes

INTERFACE:

```
      subroutine dh(  dtsub,    sal1d,    tiz
     $              ,  tbot,       hi,       hs,   fbot
     $              ,  fnet,    condb,      flh,     ni
     $              ,  dhib,     dhit,      dhs,   subi
     $              ,  subs,     dhif,     dhsf,     qi
     $              ,  focn, i,j )
```

DESCRIPTION:

Computes the thickness changes at the top and bottom and adjusts layer energy of melt; does not allow h less than 0.
Focn= actual flux of heat from the ocean layer under sea ice (equal to fbot unless all the ice melts away); compensates for rare case of melting entire slab through

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
use ice_state
use ice_diagnostics
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

```
 real (kind=dbl_kind), intent(in) ::
&   dtsub               ! timestep
&,  sal1d   (nmax+1)    ! ice salinity                          (ppt)
&,  tiz   (0:nmax)      ! snow/ice internal temp                  (C)
&,  Tbot                ! ice bottom in                           (C)
&,  hi                  ! initial ice thickness                   (m)
&,  hs                  ! initial snow thickness                  (m)
&,  fbot                ! flx from ocean, potent.              (W/m**2)
&,  fnet                ! net flx at top srf incl. cond. flx   (W/m**2)
&,  condb               ! cond. flx at bot.                    (W/m**2)
&,  flh                 ! latent heat flx                      (w/m**2)

 integer (kind=int_kind), intent(in) ::
&   ni  ! number of layers
&,  i,j ! grid location for debugging

 ! thickness changes from grow/melt (default) or sublimate/flooding
 real (kind=dbl_kind), intent(out) ::
&   dhib                ! ice bot, dhib<0 if melt                 (m)
&,  dhit                ! ice top, dhit<=0                        (m)
&,  dhs                 ! snow top, dhit<=0                       (m)
&,  subi                ! ice top, subi<0 if sublimating          (m)
&,  subs                ! snow, subs<0 if sublimating             (m)
&,  dhif                ! ice top from flooding, dhif>0           (m)
&,  dhsf                ! snow from flooding, dhsf<0              (m)
&,  qi(nmax)            ! energy of melt of ice per unit vol. (J/m**3)
&,  focn                ! actual flx of heat used from ocn    (w/m**2)
```

4.7.2 freeboard - freeboard flooding adjustment: snow-ice formation

INTERFACE:

```
subroutine freeboard(hs,hi,dhs,qs,dhsf,dhif,qiflood)
```

DESCRIPTION:

Freeboard adjustment due to flooding ... snow-ice formation

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &    hs  ! initial snow thickness                    (m)
     &,   hi  ! initial ice thickness                     (m)
     &,   dhs ! snow top, dhit<=0                         (m)
     &,   qs  ! energy of melt of snow per unit vol. (J/m**3)

      real (kind=dbl_kind), intent(out) ::
     &    dhsf ! snow from flooding, dhsf<0                (m)
     &,   dhif ! ice top from flooding, dhif>0             (m)
     &,   qiflood ! energy of melt of flooded ice    (W/m**2)
```

4.7.3 srfsub - computes sea ice and snow thickness changes

INTERFACE:

```
      subroutine srfsub(  qi,   qs, delti, delts,    ni,
     $                  subi, subs,  etop,  enet )
```

DESCRIPTION:

Compute the sea ice and snow thickness changes from sublimation/condensation

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   qi (1:nmax), qs      ! energy of melt of ice/snow per vol  (J/m**3)
     &,  delti(nmax), delts   ! thickness of ice/snow layer              (m)

      integer (kind=int_kind), intent(in) :: ni  ! number of layers

      real (kind=dbl_kind), intent(out) ::
     &   subi, subs  ! subl/cond. amount for ice/snow           (m)

      real (kind=dbl_kind), intent(inout) ::
     &   etop                ! energy avail to sub/cond ice/snow   (J/m**2)
     &,  enet                ! energy needed to melt all ice/snow  (J/m**2)
```

4.7.4 srfmelt - melt snow and ice from the top

INTERFACE:

```
      subroutine srfmelt(   qi,   qs, delti, delts, ni,
     $                   dhit,  dhs,  etop  )
```

DESCRIPTION:

Melt ice/snow from the top srf

REVISION HISTORY:

```
   author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   qi (1:nmax), qs      ! energy of melt of ice/snow per vol (J/m**3)
     &,  delti(nmax), delts   ! thickness of ice/snow layer             (m)

      integer (kind=int_kind), intent(in) :: ni    ! number of layers

      real (kind=dbl_kind), intent(out) ::
     &   dhit                 ! ice thickness change                   (m)
     &,  dhs                  ! snow thickness change                  (m)

      real (kind=dbl_kind), intent(inout) ::
     &   etop                 ! energy avail to melt ice and snow  (J/m**2)
```

4.7.5 botmelt - melt from bottom

INTERFACE:

```
      subroutine botmelt(   qi,   qs, delti, delts, ni,
     $                   dhib,  dhs,  ebot  )
```

DESCRIPTION:

Melt from bottom

REVISION HISTORY:

```
   author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   qi (1:nmax), qs      ! energy of melt of ice/snow per vol (J/m**3)
     &,  delti(nmax), delts   ! thickness of ice/snow layer             (m)

      integer (kind=int_kind), intent(in) :: ni    ! number of layers

      real (kind=dbl_kind), intent(out) :: dhib  ! ice thickness change (m)

      real (kind=dbl_kind), intent(inout) ::
     &   dhs                  ! snow thickness change                  (m)
     &,  ebot                 ! energy avail to melt ice and snow  (J/m**2)
```

4.7.6 adjust - adjusts temperature profile to changing layer spacing

INTERFACE:

```
subroutine adjust(hi0,dhib,dhit,dhif,dhsf,qiflood,qigrow,ni,qi_tw)
```

DESCRIPTION:

Adjusts temperature profile to account for changing the layer spacing due to growth/melt (incl. subl/cond, flooding). At start the energy of melting was computed after updating tiz from the heat equation. hi is the thickness prior to changes from dhib and dhit; hi_tw is the thickness after making these changes; dhib is less than 0 if there is melt at the bottom; dhit is less than 0 if there is melt at the top; generally _tw is a suffix to label the adjusted variables.

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   hi0                 ! initial ice thickness                   (m)
     &,  dhib                ! ice bot, dhib<0 if melt                 (m)
     &,  dhit                ! ice top, dhit<=0                        (m)
     &,  dhif                ! ice top from flooding, dhif>0           (m)
     &,  dhsf                ! snow from flooding, dhsf<0              (m)
     &,  qiflood             ! qi for flooded ice                (J/m**3)
     &,  qigrow              ! qi for ice growing on bot         (J/m**3)

      integer (kind=int_kind), intent(in) :: ni

      real (kind=dbl_kind), intent(inout) ::
     &   qi_tw(nmax)         ! energy of melt of ice per unit vol. (J/m**3)
```

4.7.7 real function energ - compute energy of melting per unit volume

INTERFACE:

```
real function energ(Tmp ,sal)
```

DESCRIPTION:

Compute the energy of melting per unit volume (J/m**3) relative to melting (negative quantity)

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &    Tmp          ! midpt temperature of ice layer       (C)
     &,   sal          ! midpt salinity of ice layer        (ppt)
```

4.8 Fortran: Module Interface ice_diagnostics - diagnostic information output during run (Source File: ice_diagnostics.F)

Diagnostic information output during run

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         Bruce P. Briegleb, NCAR
```

INTERFACE:

```
module ice_diagnostics
```

USES:

```
use ice_domain
use ice_constants
```

4.8.1 runtime_diags - writes max,min,global sums to standard out

INTERFACE:

```
subroutine runtime_diags
```

DESCRIPTION:

Writes diagnostic info (max, min, global sums, etc) to standard out

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         Bruce P. Briegleb, NCAR
         Cecilia M. Bitz, UW
```

USES:

```
use ice_model_size
use ice_flux
use ice_albedo
use ice_mpi_internal
use ice_history
use ice_grid
use ice_calendar
use ice_state
use ice_dyn_evp
use shr_sys_mod, only : shr_sys_flush
```

INPUT/OUTPUT PARAMETERS:

4.8.2 init_mass_diags - computes global combined ice and snow mass sum

INTERFACE:

```
subroutine init_mass_diags
```

DESCRIPTION:

Computes global combined ice and snow mass sum

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_mpi_internal
use ice_grid
use ice_state
```

INPUT/OUTPUT PARAMETERS:

4.8.3 init_diags - initialize diagnostic output

INTERFACE:

```
subroutine init_diags
```

DESCRIPTION:

Initialize diagnostic output

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

4.8.4 print_state - print ice state for specified grid point

INTERFACE:

```
subroutine print_state(plabel,i,j)
```

DESCRIPTION:

This routine is useful for debugging call to it should be inserted in the form (after thermo, for example)
do j=jlo,jhi
-do i=ilo,ihi
—call to_column(i,j)
—plabel = 'post thermo'
—if (istep1.ge.check_step.and.i.eq.ip.and.j.eq.jp
——.and.my_task.eq.mtask)
—call print_state(plabel,i,j)
-enddo
enddo

'use ice_diagnostics' may need to be inserted also, and
'use ice_calendar' if it is not already being used

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_model_size
use ice_kinds_mod
use ice_calendar
use ice_state
use ice_grid
use ice_itd
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

```
character (len=20), intent(in) :: plabel
integer (kind=int_kind), intent(in) :: i,j
```

4.9 Fortran: Module Interface ice_domain - sets array sizes for local subdomain and parallel info (Source File: ice_domain.F)

Sets array sizes for local subdomain and related parallel processing info. Code originally based on domain.F in POP

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_domain
```

USES:

```
use ice_kinds_mod
use ice_model_size
```

4.10 Fortran: Module Interface ice_dyn_evp - elastic-viscous-plastic sea ice dynamics model (Source File: ice_dyn_evp.F)

Elastic-viscous-plastic sea ice dynamics model
Computes ice velocity

See E. C. Hunke and J. K. Dukowicz. An elastic-viscous-plastic model
for sea ice dynamics. 1997, J. Phys. Oceanogr., Vol 27, 1849–1867.

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

INTERFACE:

```
module ice_dyn_evp
```

USES:

```
use ice_kinds_mod
use ice_domain
use ice_grid
use ice_constants
```

4.10.1 evp - elastic-viscous-plastic dynamics driver

INTERFACE:

```
subroutine evp(kstrngth)
```

DESCRIPTION:

Elastic-viscous-plastic dynamics driver

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

```
use ice_timers
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) ::
&   kstrngth
```

4.10.2 init_evp - initialize parameters needed for evp dynamics

INTERFACE:

```
subroutine init_evp
```

DESCRIPTION:

Initialize parameters and variables needed for the evp dynamics

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

```
use ice_calendar
```

INPUT/OUTPUT PARAMETERS:

4.10.3 evp_prep - compute quantities needed for stress tensor and mom eqns

INTERFACE:

```
subroutine evp_prep(kstrngth)
```

DESCRIPTION:

Computes quantities needed in the stress tensor (sigma)
and momentum (u) equations, but which do not change during
the thermodynamics/transport time step:

–wind stress shift to U grid,
–ice mass and ice extent masks,
–pressure (strength), and part of the forcing stresses
initializes ice velocity for new points to ocean sfc current

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

```
use ice_flux
use ice_calendar
use ice_state
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) ::
&   kstrngth
```

4.10.4 stress - computes strain rates and internal stress components

INTERFACE:

```
subroutine stress(ksub)
```

DESCRIPTION:

Computes the rates of strain and internal stress components for
each of the four corners on each T-grid cell

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: ksub  ! subcycling step
```

4.10.5 stepu - surface stresses and integrates mom eqn for u,v

INTERFACE:

```
subroutine stepu
```

DESCRIPTION:

Calculation of the surface stresses
Integration of the momentum equation to find velocity (u,v)

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

```
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

4.10.6 evp_finish - calculates ice-ocean stress

INTERFACE:

```
subroutine evp_finish
```

DESCRIPTION:

Calculation of the ice-ocean stress
...the sign will be reversed later...

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

```
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

4.10.7 principal_stress - computes principal stress for yield curve

INTERFACE:

```
subroutine principal_stress
```

DESCRIPTION:

Computes principal stresses for comparison with the theoretical yield curve; northeast values

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

USES:

INPUT/OUTPUT PARAMETERS:

4.11 Fortran: Module Interface ice_flux - flux variable declarations: from, to coupler and internal (Source File: ice_flux.F)

Flux variable declarations; these include fields sent from the coupler ("in"), sent to the coupler ("out"), and used internally ("internal")

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_flux
```

USES:

```
use ice_kinds_mod
use ice_domain
use ice_constants
```

4.11.1 init_flux_atm - initialize all atmospheric fluxes sent to coupler

INTERFACE:

```
subroutine init_flux_atm(i,j)
```

DESCRIPTION:

Initialize all fluxes sent to coupler for use by the atm model and a few state quantities

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer i,j
```

4.11.2 init_flux_ocn - initialize ocean fluxes sent to coupler

INTERFACE:

```
subroutine init_flux_ocn(i,j)
```

DESCRIPTION:

Initialize fluxes sent to coupler for use by the ocean model

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer i,j
```

4.11.3 merge_fluxes - aggregate flux information over ITD

INTERFACE:

```
subroutine merge_fluxes(i,j,nc,frshn,evapn,
&      fshn,flhn,flwupn,swbot,Focn,Trefn,strxn,stryn)
```

DESCRIPTION:

Aggregates flux information from all ice thickness categories

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_state
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i,j,nc
real (kind=dbl_kind), intent(in) ::
&    frshn       ! fresh water flux to ocean       (kg/m2/s)
&,   evapn       ! evaporation                     (kg/m2/s)
&,   fshn        ! sensible heat flx               (W/m**2)
&,   flhn        ! latent   heat flx               (W/m**2)
&,   flwupn      ! upwd lw emitted heat flx        (W/m**2)
&,   swbot       ! sw radiation through ice bot    (W/m**2)
&,   Focn        ! actual ocn/ice heat flx         (W/m**2)
&,   Trefn       ! air tmp rfrnc level             (K)
&,   strxn       ! air/ice zonal  strss,           (N/m**2)
&,   stryn       ! air/ice merdnl strss,           (N/m**2)
```

4.12 Fortran: Module Interface ice_grid - spatial grids, masks and boundary conditions (Source File: ice_grid.F)

Spatial grids, masks, and boundary conditions

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         Tony Craig, NCAR
```

INTERFACE:

```
module ice_grid
```

USES:

```
use ice_kinds_mod
use ice_constants
use ice_domain
```

4.12.1 init_grid - horizontal grid initialization

INTERFACE:

```
subroutine init_grid(oceanmixed_ice)
```

DESCRIPTION:

Horizontal grid initialization:

HTN,E = cell widths on N,E sides of T cell;
ULAT,LONG = true latitude,longitude of U points;
DX,YT,U = x,y spacing centered at T,U points.

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.12.2 popgrid - reads and sets pop displaced pole grid and land mask

INTERFACE:

```
subroutine popgrid(oceanmixed_ice)
```

DESCRIPTION:

POP displaced pole grid and land mask. Grid record number,
field and units are:
(1) ULAT (radians)
(2) ULON (radians)
(3) HTN (cm)
(4) HTE (cm)
(5) HUS (cm)
(6) HUW (cm)
(7) ANGLE (radians).

Land mask record number and field is (1) KMT.

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_read_write
```

INPUT/OUTPUT PARAMETERS:

4.12.3 columngrid - column grid and mask

INTERFACE:

```
subroutine columngrid
```

DESCRIPTION:

Column grid and mask

REVISION HISTORY:

```
author: C. M. Bitz UW, (based on rectgrid by Hunke)
```

USES:

```
use ice_model_size
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.12.4 rectgrid - regular rectangular grid and mask

INTERFACE:

```
subroutine rectgrid
```

DESCRIPTION:

Regular rectangular grid and mask

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

```
use ice_model_size
use ice_mpi_internal
```

INPUT/OUTPUT PARAMETERS:

4.12.5 makemask - makes logical land masks (T,U) and hemispheric masks

INTERFACE:

```
subroutine makemask
```

DESCRIPTION:

Sets the boundary values for the T cell land mask (hm) and makes the logical land masks for T and U cells (tmask, umask). Also creates hemisphere masks (mask-n northern, mask-s southern)

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

4.12.6 Tlatlon - initializes latitude and longitudes on T grid

INTERFACE:

```
subroutine Tlatlon
```

DESCRIPTION:

Initializes latitude and longitude on T grid

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL; code originally based on POP grid
generation routine
```

USES:

```
use ice_model_size
use ice_mpi_internal
 use ice_read_write ! if reading ULAT, ULON directly from file
```

INPUT/OUTPUT PARAMETERS:

4.12.7 t2ugrid - transfer from T-cell centers to U-cell centers

INTERFACE:

```
subroutine t2ugrid(work)
```

DESCRIPTION:

Transfer from T-cell centers to U-cell centers. Writes work into another array that has ghost cells

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: work (ilo:ihi,jlo:jhi)
```

4.12.8 to_ugrid - shift from T-cell to U-cell midpoints

INTERFACE:

```
subroutine to_ugrid(work1,work2)
```

DESCRIPTION:

Shifts quantities from the T-cell midpoint (work1) to the U-cell midpoint (work2)

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
 real (kind=dbl_kind) :: work1(imt_local,jmt_local)
&,                       work2(ilo:ihi,jlo:jhi)
```

4.12.9 u2tgrid - transfer from U-cell centers to T-cell centers

INTERFACE:

```
subroutine u2tgrid(work)
```

DESCRIPTION:

Transfer from U-cell centers to T-cell centers. Writes work into another array that has ghost cells

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: work (ilo:ihi,jlo:jhi)
```

4.12.10 to_tgrid - shifts array from U-cell to T-cell midpoints

INTERFACE:

```
subroutine to_tgrid(work1,work2)
```

DESCRIPTION:

Shifts quantities from the U-cell midpoint (work1) to the T-cell midpoint (work2)

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: work1(imt_local,jmt_local)
&,                      work2(ilo:ihi,jlo:jhi)
```

4.12.11 bound - fills ghost cells with boundary information

INTERFACE:

```
subroutine bound(work1)
```

DESCRIPTION:

Fills ghost cells with boundary information

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: work1(1)
```

4.12.12 bound_sw - fills south and west ghost cells

INTERFACE:

```
subroutine bound_sw(work1)
```

DESCRIPTION:

Fills south and west ghost cells with boundary information

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: work1(1)
```

4.12.13 bound_narr_ne - fills north and east ghost cells

INTERFACE:

```
subroutine bound_narr_ne(narrays,work1)
```

DESCRIPTION:

Fills north and east ghost cells with boundary information.
NOTE: work1 array has form (number-arrays,i-index,j-index)
for evp dynamics performance

REVISION HISTORY:

```
authors: Tony Craig, NCAR
         Elizabeth Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: narrays
real (kind=dbl_kind) :: work1(1)
```

4.12.14 bound_narr - fills north and east ghost cells with boundary info

INTERFACE:

```
subroutine bound_narr(narrays,work1)
```

DESCRIPTION:

Fills north and east ghost cells with boundary information; narr arrays at once (for performance)

REVISION HISTORY:

```
authors: Tony Craig, NCAR
         Elizabeth Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: narrays
real (kind=dbl_kind) :: work1(1)
```

4.12.15 bound_ijn - Periodic/Neumann boundary conditions

INTERFACE:

```
subroutine bound_ijn(nd,work1,north,south,east,west)
```

DESCRIPTION:

Periodic/Neumann conditions for global domain boundaries.
Assumptions: a *single* row of ghost cells (num-ghost-cells=1);
work1 array has form (i-index,j-index,number-arrays)

REVISION HISTORY:

```
authors: Tony Craig, NCAR
         Elizabeth Hunke, LANL
```

USES:

```
use ice_timers
use ice_mpi_internal
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: nd
real (kind=dbl_kind) :: work1(imt_local,jmt_local,nd)
logical north,south,east,west
```

4.12.16 bound_nij - Periodic/Neumann boundary conditions

INTERFACE:

```
subroutine bound_nij(nd,work1,north,south,east,west)
```

DESCRIPTION:

Periodic/Neumann conditions for global domain boundaries.
Assumptions: a *single* row of ghost cells (num-ghost-cells=1);
work1 array has form (number-arrays,i-index,j-index)

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

```
use ice_timers
use ice_mpi_internal
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: nd
real (kind=dbl_kind) :: work1(nd,imt_local,jmt_local)
logical north,south,east,west
```

4.13 Fortran: Module Interface ice_history - ice model history and restart files (Source File: ice_history.F)

Output files: netCDF data, Fortran unformatted dumps

REVISION HISTORY:

```
authors Tony Craig, NCAR
        Elizabeth C. Hunke, LANL
        C.M. Bitz, UW
        Bruce P. Briegleb, NCAR
```

INTERFACE:

```
module ice_history
```

USES:

```
use ice_kinds_mod
use ice_domain
use ice_read_write
```

4.13.1 init_hist - initialize history files

INTERFACE:

```
subroutine init_hist
```

DESCRIPTION:

Initialize history files

REVISION HISTORY:

```
authors Tony Craig, NCAR
        Elizabeth C. Hunke, LANL
        C.M. Bitz, UW
        Bruce P. Briegleb, NCAR
```

USES:

```
use ice_constants
use ice_calendar
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.13.2 ice_write_hist - write average ice quantities or snapshots

INTERFACE:

```
subroutine ice_write_hist
```

DESCRIPTION:

write average ice quantities or snapshots

REVISION HISTORY:

```
author:   E.C.Hunke, LANL
```

USES:

```
use ice_flux
use ice_albedo
use ice_mechred
use ice_grid
use ice_calendar
use ice_state
use ice_dyn_evp
use ice_constants
```

INPUT/OUTPUT PARAMETERS:

4.13.3 icecdf - write netCDF history file

INTERFACE:

```
subroutine icecdf
```

DESCRIPTION:

write netCDF history file

REVISION HISTORY:

```
authors:   E.C.Hunke, LANL
           Bruce P. Briegleb, NCAR
```

USES:

```
use ice_model_size
use ice_constants
use ice_mpi_internal
use ice_grid
use ice_calendar
use ice_itd, only : c_hi_range
use shr_file_mod
```

INPUT/OUTPUT PARAMETERS:

4.13.4 nf_stat_check - check error code returned from netCDF call

INTERFACE:

```
subroutine nf_stat_check (status,msg)
```

DESCRIPTION:

Check error code returned from netCDF calls
If bad, abort

REVISION HISTORY:

```
author:   ?
```

USES:

```
use shr_sys_mod, only : shr_sys_abort
include "netcdf.inc"
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: status
character (len = *), intent(in) :: msg
```

4.13.5 integer function lenstr(label) - compute length string

INTERFACE:

```
integer function lenstr(label)
```

DESCRIPTION:

Check error code returned from netCDF calls. If bad, abort

REVISION HISTORY:

```
author:   ?
```

INPUT/OUTPUT PARAMETERS:

```
character*(*) label
```

4.13.6 dumpfile - dumps all fields required for restart

INTERFACE:

```
subroutine dumpfile
```

DESCRIPTION:

Dumps all values needed for a restart

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

USES:

```
use ice_model_size
use ice_flux
use ice_grid
use ice_calendar
use ice_state
use ice_dyn_evp
use shr_file_mod
use ice_ocean
```

INPUT/OUTPUT PARAMETERS:

4.13.7 restartfile - restarts from a dumpfile

INTERFACE:

```
subroutine restartfile
```

DESCRIPTION:

Restarts from a dump

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

USES:

```
use ice_model_size
use ice_flux
use ice_mpi_internal
use ice_grid
use ice_calendar
use ice_state
use ice_dyn_evp
use ice_itd
use ice_ocean
```

INPUT/OUTPUT PARAMETERS:

4.14 Fortran: Module Interface ice_init - parameter and variable initializations (Source File: ice_init.F)

parameter and variable initializations

REVISION HISTORY:

```
authors Elizabeth C. Hunke, LANL
        C. M. Bitz
```

INTERFACE:

```
module ice_init
```

USES:

```
use ice_domain
```

4.14.1 input_data - namelist variables

INTERFACE:

```
subroutine input_data
```

DESCRIPTION:

Namelist variables, set to default values; may be altered at run time

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

USES:

```
use ice_mechred
use ice_diagnostics
use ice_history
use ice_calendar
use ice_dyn_evp
use ice_itd
use ice_transport
use ice_prescribed
use ice_ocean
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.14.2 init_state - initialize state for itd

INTERFACE:

```
subroutine init_state
```

DESCRIPTION:

Initialize state for the itd model

REVISION HISTORY:

```
author C. M. Bitz
```

USES:

```
use ice_model_size
use ice_constants
use ice_flux
use ice_vthermo
use ice_grid
use ice_state
use ice_itd
use ice_dh
```

INPUT/OUTPUT PARAMETERS:

4.14.3 init_flux - initialize fluxes exchanged with coupler

INTERFACE:

```
subroutine init_flux
```

DESCRIPTION:

Initialize all fluxes exchanged with flux coupler and some data derived fields

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
```

USES:

```
use ice_constants
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

4.14.4 setup_mpi - initialize mpi

INTERFACE:

```
subroutine setup_mpi
```

DESCRIPTION:

This routine initializes mpi for either internal parallel processing or for message passing with the coupler

REVISION HISTORY:

```
author Elizabeth C. Hunke, LANL
code originally based on POP routine
```

USES:

```
use ice_mpi_internal
use ice_coupling
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.15 Fortran: Module Interface ice_itd - initialize and redistribute ice in the ITD (Source File: ice_itd.F)

Routines to initialize the ice thickness distribution and utilities to redistribute ice among categories. These routines are not specific to a particular numerical implementation.

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669–15,677.

See Bitz, C.M., M.M. Holland, A.J. Weaver, M. Eby, 2001: Simulating the ice-thickness distribution in a climate model, J. Geophys. Res., 106, 2441–2464.

REVISION HISTORY:

```
author: C. M. Bitz, UW
code heavily modified by Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_itd
```

USES:

```
use ice_kinds_mod
use ice_model_size
use ice_constants
use ice_state
```

4.15.1 init_itd - initalize area fraction and thickness boundaries for ITD

INTERFACE:

```
subroutine init_itd
```

DESCRIPTION:

Initialize area fraction and thickness boundaries for the itd model

REVISION HISTORY:

```
authors: William H. Lipscomb, LANL
         Elizabeth C. Hunke LANL  and C. M. Bitz UW
```

USES:

```
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.15.2 aggregate - aggregate ice state over the grid

INTERFACE:

```
subroutine aggregate
```

DESCRIPTION:

Aggregate ice state over the grid

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
use ice_domain
use ice_flux
use ice_grid
```

INPUT/OUTPUT PARAMETERS:

4.15.3 aggregate_pt - aggregate a point over ITD state

INTERFACE:

```
subroutine aggregate_pt(i,j)
```

DESCRIPTION:

Aggregate ice thickness distribution state

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i, j
```

4.15.4 to_column - define column state variables for given point

INTERFACE:

```
subroutine to_column(i,j)
```

DESCRIPTION:

Define column state variables for given point (i,j)

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         and C. M. Bitz, UW
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i,j
```

4.15.5 from_column - reload state variables from given point

INTERFACE:

```
subroutine from_column(i,j)
```

DESCRIPTION:

Reload state variables for given point (i,j) from column variables

REVISION HISTORY:

```
authors: Elizabeth C. Hunke, LANL
         and C. M. Bitz, UW
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i,j
```

4.15.6 check_state - require certain fields to be monotone

INTERFACE:

```
subroutine check_state(i,j)
```

DESCRIPTION:

Insist that certain fields are monotone. Should not be necessary if all is well, but best to keep going. Model will not conserve energy and water if fields are zeroed here.

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: i,j
```

4.15.7 distr_check - enforce ice area minimum and ice thickness boundaries

INTERFACE:

```
subroutine distr_check
```

DESCRIPTION:

Force ice thickness distribution to maintain two rules
(1) the fractional area cannot be less than some limiting value
(2) each categories thickness lies within the
max and min thickness range for that category

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
use ice_flux
use ice_grid
```

INPUT/OUTPUT PARAMETERS:

4.15.8 normalize_state - normalize ice state after small areas removed

INTERFACE:

```
subroutine normalize_state(dvicen,dvices,dvsnon,dvsnos)
```

DESCRIPTION:

Normalize the state variables for the ice thickness distribution to conserve volume after removing small areas

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

```
      use ice_model_size
      use ice_mpi_internal
      use ice_grid
      use ice_calendar
```

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   dvicen(1), dvsnon(1)      ! total nuked ice/snow volumes north
     &,  dvices(1), dvsnos(1)      ! total nuked ice/snow volumes south
```

4.15.9 rebin_ice - rebins thicknesses into defined categories

INTERFACE:

```
      subroutine rebin_ice(i,j)
```

DESCRIPTION:

Rebins thicknesses into defined categories

REVISION HISTORY:

```
   author: C. M. Bitz, UW
```

USES:

```
      use ice_flux
```

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) ::
     &   i,j
```

4.15.10 movedn - moves ice from cat 2 to cat 1

INTERFACE:

```
      subroutine movedn(   Tf0
     $                  ,  ai1,   vi1,  vs1, ei1, Tsf1
     $                  ,  hi1,   hs1
     &                  , h1min,  nn1
     $                  ,  ai2,   vi2,  vs2, ei2, Tsf2
     $                  ,  hi2,   hs2
     &                  , h2min,  nn2       )
```

DESCRIPTION:

Moves ice from cat 2 to cat 1.
It is possible that the final thickness will be below the boundary of cat 1 if the ice has melted a lot, in which case it will be moved down again by a subsequent call

REVISION HISTORY:

```
   author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   Tf0          ! freezing temperature of ocean
     &,  h1min, h2min ! minimum thickness range of cat 1, 2     (m)
      integer (kind=int_kind), intent(in) ::
     &   nn1,   nn2   ! number of layers in cat 1, 2

      real (kind=dbl_kind), intent(inout) ::
     &   ai1,  ai2    ! fraction of ice
     &,  vi1,  vi2    ! volume per unit area of ice             (m)
     &,  vs1,  vs2    ! volume per unit area of snow            (m)
     &,  hi1,  hi2    ! ice thickness                           (m)
     &,  hs1,  hs2    ! snow thickness                          (m)
     &,  Tsf1, Tsf2   ! ice/snow top surf. temperature          (K)

      real (kind=dbl_kind), intent(inout), dimension (:) ::
     &   ei1          ! energy of melting of ice per layer (J/m**2)
     &,  ei2          ! energy of melting of ice per layer (J/m**2)
```

4.15.11 moveup - moves ice from cat 1 to cat 2

INTERFACE:

```
      subroutine moveup(     Tf0
     $                     ,  ai1,   vi1,  vs1, ei1, Tsf1
     $                     ,  hi1,  hs1
     &                     , h1min,  nn1
     $                     ,  ai2,   vi2,  vs2, ei2, Tsf2
     $                     ,  hi2,  hs2
     &                     , h2min,  nn2       )
```

DESCRIPTION:

Moves ice from cat 1 to 2.
It is possible that the final thickness will be above the boundary of cat 2. If the ice needs to move up again, it will do so in a subsequent call

REVISION HISTORY:

```
author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   Tf0          ! temperature of ocean
     &,  h1min, h2min ! minimum thickness range of cat 1, 2     (m)
      integer (kind=int_kind), intent(in) ::
     &   nn1,   nn2   ! number of layers in cat 1, 2

      real (kind=dbl_kind), intent(inout) ::
     &   ai1,  ai2    ! fraction of ice
     &,  vi1,  vi2    ! volume per unit area of ice             (m)
```

```
     &,  vs1,  vs2    ! volume per unit area of snow           (m)
     &,  hi1,  hi2    ! ice thickness                          (m)
     &,  hs1,  hs2    ! snow thickness                         (m)
     &,  Tsf1, Tsf2   ! ice/snow top surf. temperature         (K)
     &,  ei1(nn1)     ! energy of melting of ice per layer (J/m**2)
     &,  ei2(nn2)     ! energy of melting of ice per layer (J/m**2)
```

4.15.12 zerocat - reset category variables to zero when no ice

INTERFACE:

```
      subroutine zerocat( Tf0, ai, vi,  vs, ei
     $                  , Tsf
     &                  , hs, hi
     &                  , hmin,  nn )
```

DESCRIPTION:

Reset category variables in the case of no ice

REVISION HISTORY:

```
   author: C. M. Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::  Tf0    ! freezing temperature
      integer (kind=int_kind), intent(in) :: nn   ! number of layers in cat
      real (kind=dbl_kind), intent(in) :: hmin

      real (kind=dbl_kind), intent(inout) ::
     &   ai           ! fraction of ice
     &,  vi           ! volume per unit area of ice            (m)
     &,  vs           ! volume per unit area of snow           (m)
     &,  hi           ! ice thickness                          (m)
     &,  hs           ! snow thickness                         (m)
     &,  Tsf          ! ice/snow top surf. temperature         (K)
     &,  ei (nn)      ! energy of melting of ice per layer (J/m**2)
```

4.16 Fortran: Module Interface ice_itd_linear - linear remapping scheme for ITD (Source File: ice_itd_linear.F)

Linear remapping scheme for the ice thickness distribution
See Lipscomb, W. H. Remapping the thickness distribution of sea ice. 2001, J. Geophys. Res., Vol 106, 13989–14000.

REVISION HISTORY:

```
   authors: William H. Lipscomb, LANL
            Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_itd_linear
```

USES:

```
use ice_model_size
use ice_kinds_mod
use ice_constants
use ice_state
use ice_itd
use shr_sys_mod, only : shr_sys_abort
```

4.16.1 linear_itd - ITD scheme that shifts ice among categories

INTERFACE:

```
subroutine linear_itd(i,j,dhin,hin_old,hin,hsn)
```

DESCRIPTION:

Ice thickness distribution scheme that shifts ice among categories.

The default scheme is linear remapping, which works as follows. See Lipscomb (2001) for more details.

Using the thermodynamic "velocities", interpolate to find the velocities in thickness space at the category boundaries, and compute the new locations of the boundaries. Then for each category, compute the thickness distribution function, g(h), between hL and hR, the left and right boundaries of the category. Assume g(h) is a linear polynomial that satisfies two conditions:

(1) The ice area implied by g(h) equals ain(n).
(2) The ice volume implied by g(h) equals ain(n)*hice(n).

Once g(h) is computed, compute the ice area and volume lying between the initial and new boundaries, and transfer this area and volume to the neighboring cell, thus restoring the initial boundary.

REVISION HISTORY:

```
authors: William H. Lipscomb, LANL
         Elizabeth C. Hunke, LANL
```

USES:

```
use ice_domain
use ice_diagnostics
use ice_calendar
```

INPUT/OUTPUT PARAMETERS:

```
 integer (kind=int_kind), intent(in) ::
&  i,j                ! grid indices

 real (kind=dbl_kind), intent(in) ::
&   dhin(ncat)        ! thickness change for remapping    (m)
&,  hin_old(ncat)     ! starting value of hin             (m)

 real (kind=dbl_kind), intent(inout) ::
&   hin(ncat)         ! ice thickness for each cat        (m)
&,  hsn(ncat)         ! snow thickness for each cat       (m)
```

4.16.2 fit_line - fit g(h) with a line using area, volume constraints

INTERFACE:

```
      subroutine fit_line (n, hin, Hb, g0, g1, hL, hR, i,j)
```

DESCRIPTION:

Fit g(h) with a line, satisfying area and volume constraints. To reduce rounding errors caused by large values of g0 and g1, we compute g(eta), where eta = h - hL.

REVISION HISTORY:

```
   authors: William H. Lipscomb, LANL
            Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) ::
     &   n               ! thickness category for which g(h) computed

      real (kind=dbl_kind), intent(in) ::
     &   hin(ncat)       ! ice thickness for each cat        (m)
     &,  Hb(0:ncat)      ! current category boundaries

      real (kind=dbl_kind), intent(out) ::
     &   g0, g1          ! coefficients in linear equation for g(eta)
     &,  hL              ! min value of range over which g(h) > 0
     &,  hR              ! max value of range over which g(h) > 0

      integer (kind=int_kind), intent(in) ::
     &   i,j
```

4.16.3 shift_ice - shift ice across category boundaries

INTERFACE:

```
      subroutine shift_ice (donor, daice, dvice, hin, i,j)
```

DESCRIPTION:

Shift ice across category boundaries, conserving area, volume, and energy.

REVISION HISTORY:

```
   authors: William H. Lipscomb, LANL
            Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
 integer (kind=int_kind), intent(in) ::
&   donor(ncat-1)      ! donor category index

 real (kind=dbl_kind), intent(inout) ::
&   daice(ncat-1)      ! ice area transferred across boundary
&,  dvice(ncat-1)      ! ice volume transferred across boundary

 real (kind=dbl_kind), intent(in) ::
&   hin(ncat)          ! ice thickness for each cat        (m)

 integer (kind=int_kind), intent(in) ::
&   i,j
```

4.17 Fortran: Module Interface ice_kinds_mod - defines variable precision (Source File: ice_kinds_mod.F)

Defines variable precision for all common data types
Code originally based on kinds_mod.F in POP

REVISION HISTORY:

```
author: Elizabeth C. Hunke
        Fluid Dynamics Group, Los Alamos National Laboratory
```

INTERFACE:

```
module ice_kinds_mod
```

USES:

4.18 Fortran: Module Interface ice_mechred - computes mech redistribution and strength for ITD (Source File: ice_mechred.F)

Ice thickness distribution model with multi-layer thermodynamics. Routines to compute mechanical redistribution and strength.

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669-15,677.

See Bitz, C.M., M.M. Holland, A.J. Weaver, M. Eby, 2001: Simulating the ice-thickness distribution in a climate model, J. Geophys. Res., 106, 2441–2464.

REVISION HISTORY:

```
author: C. M. Bitz
```

INTERFACE:

```
module ice_mechred
```

USES:

```
use ice_model_size
use ice_constants
use ice_state
use ice_itd
use ice_dyn_evp
```

4.18.1 init_mechred - initialize constants for ridging

INTERFACE:

```
subroutine init_mechred
```

DESCRIPTION:

Initialize constants for ridging

REVISION HISTORY:

```
author: C.M.Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

4.18.2 mechanical_redistr - driver for ridging

INTERFACE:

```
subroutine mechanical_redistr
```

DESCRIPTION:

This routine works with ncat=1
In fact, it will provide a source of open water from shear deformation. See Stern et al, 1995 for information about how and why this is done

Keeps track of heat and fresh water flx to ocn
if snow on ice that ridges is thrown into the ocean

REVISION HISTORY:

```
author: C.M.Bitz, UW
```

USES:

```
use ice_domain
use ice_flux
use ice_grid
use ice_timers
use ice_calendar, only: dt
```

INPUT/OUTPUT PARAMETERS:

4.18.3 ridge - ridge ice due to convergence or shear deformation

INTERFACE:

```
      subroutine ridge( dt1, closng, epsi, delta_local, Tf1,
     $                  Fhnet1, Fresh1, Gamm, H2Gamm)
```

DESCRIPTION:

Ridge ice due to convergence or shear deformation
note the timestep is dt1 for ridging among the cats
however will subcycle with timestep dtsub if a cat
would run out of ice if the timestep were greater than dtsub

Throw snow on ice that ridges into ocean if snow_into_ocn=.true.

REVISION HISTORY:

```
  author: C.M.Bitz, UW
```

USES:

```
      use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   dt1
     &,  delta_local         ! Delta from dynamics routine
     &,  Tf1                 ! ocean freezing temperature

      real (kind=dbl_kind), intent(out) ::
     &   Fhnet1              ! heat given to ocean from snow             (N/m)
     &,  Fresh1              ! water given to ocean from snow              (m)
     &,  H2Gamm  (ncat,ncat) ! H*volume frac from cat i that goes into j

      real (kind=dbl_kind), intent(inout) ::
     &   Gamm    (ncat,ncat) ! area frac from cat i that goes into j
     &,  epsi                ! actual divergence computed from transport (1/s)
     &,  closng              ! closing from Flato & Hibler Eq. 9         (1/s)
```

4.18.4 ridging_mode - compute W and Wa factors

INTERFACE:

```
      subroutine ridging_mode( W, Wa, Gamm )
```

DESCRIPTION:

Compute W and Wa, factors for ridging mode and participation function

REVISION HISTORY:

```
  author: C.M.Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &  Gamm    (ncat,ncat) ! area frac from cat i that goes into j

      real (kind=dbl_kind), intent(out) ::
     &   W       (0:ncat)   ! W factor, geometric comp. of ridging mode
     &,  Wa      (0:ncat)   ! part of W from participation func.
```

4.18.5 ridge_matrices - compute Gamma factors

INTERFACE:

```
      subroutine ridge_matrices( hin, Hmean, Gamm, HGamm, H2Gamm )
```

DESCRIPTION:

Compute Gamm,HGamm,H2Gamm for ridge and strength for true thicknesses

REVISION HISTORY:

```
  author: C.M.Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(out) ::
     &   hin      (ncat)      ! ice thickness                            (m)
     &,  Hmean    (ncat)      ! mean thickness of ridg ice from cat nc   (m)
     &,  Gamm     (ncat,ncat) ! area frac from cat i that goes into j
     &,  HGamm    (ncat,ncat) ! volume frac from cat i that goes into j
     &,  H2Gamm   (ncat,ncat) ! H*volume frac from cat i that goes into j
       ! HGamm is used in ridge and H2Gamm is used in roth_strength
```

4.18.6 comp_matrices - compute Gamm functions

INTERFACE:

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

DESCRIPTION:

Compute Gamm, HGamm, H2Gamm
HGamm is used in ridge and H2Gamm is used in roth-strength
Gamm(nc,k) distributes ice that participates in ridging from cat nc into a distrib. of cats k

n1 is the smallest k with nonzero fract
n2 is the largest k with nonzero fract

ice that partic. has thickness hin(nc) and it ridges up to linear distrib. between 2*hin(nc) and 2*sqrt(cK*hin(nc))

REVISION HISTORY:

```
author: C.M.Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      logical (kind=log_kind), intent(in) ::
     &   rowflg(ncat)     ! true if cat has nonzero fractional area
      real (kind=dbl_kind), intent(in) ::
     &   hin      (ncat)      ! ice thickness                              (m)

      real (kind=dbl_kind), intent(out) ::
     &   Hmean    (ncat)      ! mean thickness of ridg ice from cat nc  (m)
     &,  Gamm     (ncat,ncat) ! area frac from cat i that goes into j
     &,  HGamm    (ncat,ncat) ! volume frac from cat i that goes into j
     &,  H2Gamm   (ncat,ncat) ! H*volume frac from cat i that goes into j
```

4.18.7 real function roth_strength - Rothrock ice strength

INTERFACE:

```
      real function roth_strength( Gamm ,H2Gamm )
```

DESCRIPTION:

Compute the ice strength based on Rothrock, 1975 and also see FH95
does not make sense to do this unless the ice that participates in
ridging is well resolved – must have about 5 categories, 10 would be better

REVISION HISTORY:

```
author: C.M.Bitz, UW
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(in) ::
     &   Gamm     (ncat,ncat) ! area frac from cat i that goes into j
     &,  H2Gamm   (ncat,ncat) ! H*volume frac from cat i that goes into j
```

4.19 Fortran: Module Interface ice_model_size - defines global domain size, category and layer number (Source File: ice_model_size.F)

Defines the global domain size and number of categories and layers.
Code originally based on model_size.F in POP

REVISION HISTORY:

```
author Elizabeth C. Hunke
       Fluid Dynamics Group, Los Alamos National Laboratory
```

INTERFACE:

```
      module ice_model_size
```

USES:

```
      use ice_kinds_mod
```

4.20 Fortran: Module Interface ice_mpi_internal - parameters and common blocks for MPI internal to ice model (Source File: ice_mpi_internal.F)

Parameters and commons blocks for MPI parallelization internal to ice model

REVISION HISTORY:

```
authors: Tony Craig, NCAR
         Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_mpi_internal
```

USES:

```
use ice_kinds_mod
use ice_domain
```

4.20.1 ice_global_real_minmax - computes global min/max and prints

INTERFACE:

```
subroutine ice_global_real_minmax(nc,work,string)
```

DESCRIPTION:

Determines and writes both minimum and maximum over global grid

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nc
real (kind=dbl_kind), intent(in) :: work(nc)
character (len=8), intent(in) :: string
```

4.20.2 real function ice_global_real_minval - computes global min

INTERFACE:

```
real function ice_global_real_minval(nc,work)
```

DESCRIPTION:

Computes minimum over the global grid

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nc
real (kind=dbl_kind), intent(in) :: work(nc)
```

4.20.3 real function ice_global_real_maxval - computes global max

INTERFACE:

```
real function ice_global_real_maxval(nc,work)
```

DESCRIPTION:

Computes maximum over the global grid

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nc
real (kind=dbl_kind), intent(in) :: work(nc)
```

4.20.4 real function ice_global_real_sum - sums input array over global grid

INTERFACE:

```
real function ice_global_real_sum(nc,work)
```

DESCRIPTION:

Sums given array over the global grid

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nc
real (kind=dbl_kind), intent(in) :: work(nc)
```

4.20.5 ice_bcast_logical - broadcasts scalar logical to all processors

INTERFACE:

```
subroutine ice_bcast_logical(logval)
```

DESCRIPTION:

Broadcasts a scalar logical value to all processors

REVISION HISTORY:

```
   author: Julie Schramm, NCAR
           Added 22 Oct 2001
```

USES:

INPUT/OUTPUT PARAMETERS:

```
       implicit none

       logical (kind=log_kind), intent(inout) :: logval
```

4.20.6 ice_bcast_char - broadcasts scalar character to all processors

INTERFACE:

```
       subroutine ice_bcast_char(charval)
```

DESCRIPTION:

Broadcasts a scalar character value to all processors

REVISION HISTORY:

```
   author: Julie Schramm, NCAR
           Added 22 Oct 2001
```

USES:

INPUT/OUTPUT PARAMETERS:

```
       implicit none
       character (*), intent(inout) :: charval
```

4.20.7 ice_bcast_rscalar - broadcasts real scalar all processors

INTERFACE:

```
       subroutine ice_bcast_rscalar(val)
```

DESCRIPTION:

Broadcasts a real scalar character value to all processors

REVISION HISTORY:

```
   author: Julie Schramm, NCAR
           Added 22 Oct 2001
```

USES:

INPUT/OUTPUT PARAMETERS:

```
       real (kind=dbl_kind), intent(inout) :: val
```

4.20.8 ice_bcast_iscalar - broadcasts integer scalar all processors

INTERFACE:

```
subroutine ice_bcast_iscalar(ival)
```

DESCRIPTION:

Broadcasts an integer scalar character value to all processors

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(inout) :: ival
```

4.20.9 global_scatter - scatters global to distributed array, adding ghost cells

INTERFACE:

```
subroutine global_scatter(workg,work)
```

DESCRIPTION:

Scatters a global array and adds ghost cells to create a distributed array

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

```
use ice_model_size
use ice_constants
```

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: workg(imt_global,jmt_global)
&,                      work(ilo:ihi,jlo:jhi)
```

4.20.10 global_gather - gathers distributed array into global array

INTERFACE:

```
subroutine global_gather(workg,work)
```

DESCRIPTION:

Gathers a distributed array and strips off ghost cells to create a local array with global dimensions

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

```
use ice_model_size
use ice_constants
```

INPUT/OUTPUT PARAMETERS:

```
real (kind=dbl_kind) :: workg(imt_global,jmt_global)
&,                      work(ilo:ihi,jlo:jhi)
```

4.20.11 get_sum - computes weighted sum over global grid

INTERFACE:

```
subroutine get_sum(flag,work1,work2,work3,gsum)
```

DESCRIPTION:

Computes a (weighted) sum over the global grid; if flag = 1 then work1 is weighted by work2 before being added to work3

REVISION HISTORY:

```
author: Elizabeth C. Hunke, LANL
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: flag

real (kind=dbl_kind), intent(in) ::
&   work1(ilo:ihi,jlo:jhi)
&,  work2(ilo:ihi,jlo:jhi)

real (kind=dbl_kind), intent(in) ::
&   work3(imt_local,jmt_local)

real (kind=dbl_kind), intent(out) ::
&   gsum
```

4.20.12 end_run - ends run

INTERFACE:

```
subroutine end_run
```

DESCRIPTION:

Ends run by calling MPI_FINALIZE.

REVISION HISTORY:

```
author: ?
```

USES:

INPUT/OUTPUT PARAMETERS:

```
#ifdef _MPI
      call MPI_FINALIZE(ierr)
#endif
```

4.21 Fortran: Module Interface ice_ocean - ocean mixed layer internal to sea ice model (Source File: ice_ocean.F)

Ocean mixed layer calculation (internal to sea ice model).
Open ocean has stability based flux calculations for uncoupled runs. Uses monthly mean ocean forcing data from external file to drive the ocean mixed layer. Initial sst either from the ice restart file or set to 1 Jan from external file data./initial

List of ocean forcing fields: Note that order is important!
(order is determined by field list in vname).
For ocean mixed layer——————————units
1 sst——temperature——————————(C)
2 sss——salinity——————————(ppt)
3 hbl——depth——————————(m)
4 u———surface u current——————(m/s)
5 v———surface v current——————(m/s)
6 dhdx—surface tilt x direction———(m/m)
7 dhdy—surface tilt y direction———(m/m)
8 qdp——ocean sub-mixed layer heat flux——-(W/m2)

REVISION HISTORY:

```
original authors John Weatherly, C.M. Bitz,
                 Elizabeth C. Hunke, Bruce P. Briegleb
significantly modified by  Bruce P. Briegleb
```

INTERFACE:

```
      module ice_ocean
```

USES:

```
      use ice_flux
      use ice_calendar
      use ice_prnpnt
```

4.21.1 init_oceanmixed_ice - initialize mixed layer forcing data

INTERFACE:

```
      subroutine init_oceanmixed_ice
```

DESCRIPTION:

Initialize ocean forcing data for the mixed layer. Check netCDF
file for correct dimensions, fields and other attributes, and then read in all 12 months of forcing data.

Assumes 12 months of ocean forcing data, ordered from mean January to mean December; dates of these months are ignored.

Note the assumption that if lat/lon dimensions of the netCDF files are identical to those in the code, then the grids of the netCDF file and the code are also.

REVISION HISTORY:

```
author  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
include "netcdf.inc"
```

INPUT/OUTPUT PARAMETERS:

4.21.2 time_intrplt_ocean_forcing - time interpolates ocean forcing

INTERFACE:

```
subroutine time_intrplt_ocean_forcing
```

DESCRIPTION:

Finds two months of ocean forcing data that bracket current date (for which year is ignored), and interpolate linearly in time between those two values.

REVISION HISTORY:

```
author  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_mpi_internal
use ice_grid
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
include "netcdf.inc"
```

INPUT/OUTPUT PARAMETERS:

4.21.3 set_oceanmixed_ice - set sst and frzmlt

INTERFACE:

```
subroutine set_oceanmixed_ice
```

DESCRIPTION:

Sets sst and frzmlt with ocean mixed layer values

REVISION HISTORY:

```
   author  Bruce P. Briegleb, NCAR
```

USES:

```
      use shr_sys_mod, only : shr_sys_flush
```

INPUT/OUTPUT PARAMETERS:

4.21.4 compute_oceanmixed_ice - update ocean state fields

INTERFACE:

```
      subroutine compute_oceanmixed_ice
```

DESCRIPTION:

Update sst and other ocean state fields

REVISION HISTORY:

```
   author  Bruce P. Briegleb, NCAR
```

USES:

```
      use shr_sys_mod, only : shr_sys_flush
```

INPUT/OUTPUT PARAMETERS:

4.21.5 mixed_layer(i,j) - computes open ocean fluxes

INTERFACE:

```
      subroutine mixed_layer(i,j)
```

DESCRIPTION:

Calculate flux exchange over open ocean and update mixed layer sst

REVISION HISTORY:

```
   author  Elizabeth Hunke, LANL, and Bruce P. Briegleb, NCAR
```

USES:

```
      use ice_grid
      use ice_atmo
      use ice_constants, only : TTTocn, qqqocn
```

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: i,j
```

4.21.6 ocnheat(i,j, delt, delq) - update ocean fields

INTERFACE:

```
subroutine ocnheat(i,j, delt, delq)
```

DESCRIPTION:

Update sst and freezing/melting potential; set other ocean fields from forcing data; thermal effect of ice/ocn heat will be included in the sst change after this routine is called.

REVISION HISTORY:

```
Modified by Bruce P. Briegleb
```

USES:

```
use ice_constants
use ice_state
use ice_albedo
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: i,j

real (kind=dbl_kind), intent(in) ::
&   delt   ! potential T difference   (K)
&,  delq   ! humidity difference      (kg/kg)
```

4.22 Fortran: Module Interface ice_prescribed - Prescribed Ice (Source File: ice_prescribed.F)

Prescribed ice.

Set ice concentration with prescribed data from netCDF data file; then internally prescribe ice thickness, temperature, snow cover, layer linear temperature profile; finally, compute air/ice fluxes, but no other ice thermo or dynamics; set ice/ocean fluxes to zero.

REVISION HISTORY:

```
author: Bruce P. Briegleb, NCAR
```

INTERFACE:

```
module ice_prescribed
```

USES:

```
use ice_model_size
use ice_constants
use ice_domain
use ice_grid
use ice_calendar
```

4.22.1 init_prescribed_ice - initialize prescribed ice data

INTERFACE:

```
subroutine init_prescribed_ice
```

DESCRIPTION:

Initialize climatological ice concentration data by checking existing netCDF file for correct dimensions, fields and other attributes, and then reading in first 12 months of ice cover data for climatology case and 2 months (12 and 13) for non-climatology.

Note the assumption that if lat/lon dimensions of the netCDF files are identical to those in the code, then the grids of the netCDF file and the code are also.

REVISION HISTORY:

```
author:  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.22.2 read_prescribed_ice - read in two ice concentration fields

INTERFACE:

```
subroutine read_prescribed_ice
```

DESCRIPTION:

Read in two ice concentration fields that bracket current date, and interpolate linearlly in time between those two values. Make sure final ice concentration exceeds a small constant value.

No time extrapolation allowed; if date out of range, exit. Data may have values ¡ 0 or ¿ 1; limit 1.0e-4 to 1 after time interpolation.

REVISION HISTORY:

```
author:  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.22.3 read_prescribed_ice_climate - read two climate ice concentrations

INTERFACE:

```
subroutine read_prescribed_ice_climate
```

DESCRIPTION:

Finds two months of climatological ice concentration that bracket current date (for which year is ignored), and interpolate linearlly in time between those two values. Final time inerpolated ice concentration is limited to the range 1.0e-4 to 1.

REVISION HISTORY:

```
author:  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_mpi_internal
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.22.4 set_prescribed_ice_state - set prescribed ice state

INTERFACE:

```
subroutine set_prescribed_ice_state
```

DESCRIPTION:

Set prescribed ice state using input ice concentration; set snow to specified value; set surface ice temperature to atmospheric value; use linear temperature gradient in ice to ocean temperature.

REVISION HISTORY:

```
author:  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_flux
use ice_vthermo
use ice_state
use ice_itd
use ice_dh
use shr_sys_mod, only : shr_sys_flush, shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.22.5 set_prescribed_ice_flux - Set non-computed fluxes to zero

INTERFACE:

```
subroutine set_prescribed_ice_flux
```

DESCRIPTION:

Set non-computed fluxes for prescribed ice model to zero

REVISION HISTORY:

```
author: Bruce P. Briegleb, NCAR
```

USES:

```
use ice_dyn_evp
use ice_flux
```

INPUT/OUTPUT PARAMETERS:

4.22.6 exit_prescribed_ice - close netCDF file for prescribed ice

INTERFACE:

```
subroutine exit_prescribed_ice
```

DESCRIPTION:

Close netCDF file for prescribed ice

REVISION HISTORY:

```
author: Bruce P. Briegleb, NCAR
```

INPUT/OUTPUT PARAMETERS:

4.23 Fortran: Module Interface ice_prnpnt - diagnostic grid point info during execution (Source File: ice_prnpnt.F)

This module defines the grid points desired for printout of atmospheric and ocean inputs and ice model outputs. Data for each point is written to a separate file as ascii tables. The specific points are set by lat/lons (-90 to 90 lat and -180 to +180 lon) in a data statement. A descriptive label is useful to distinguish points. The code chooses the nearest model grid point to the specified lat/lon. Points for task 0 are written in the ice model execution directory; all other task points are written one directory up (on IBM).

REVISION HISTORY:

```
author  Bruce P. Briegleb, NCAR
```

INTERFACE:

```
module ice_prnpnt
```

USES:

```
use ice_domain
use ice_constants
use ice_calendar
use shr_sys_mod, only : shr_sys_flush
```

4.23.1 intpnt - initializes printout of grid point data

INTERFACE:

```
subroutine intpnt
```

DESCRIPTION:

Initialize printout of points by finding tasks for requested points, and defining file names for each point. Prints out description of points and tasks to log file.

REVISION HISTORY:

```
author  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_grid
use ice_mpi_internal
```

INPUT/OUTPUT PARAMETERS:

4.23.2 diags_cpl_in - grid point printout of input info from coupler

INTERFACE:

```
subroutine diags_cpl_in
```

DESCRIPTION:

Printout atmospheric forcing data for selected grid points as received from the coupler.

REVISION HISTORY:

```
author  Bruce P. Briegleb, NCAR
```

USES:

```
use ice_flux
use ice_calendar
```

INPUT/OUTPUT PARAMETERS:

4.23.3 diags_cpl_out - grid point printout of info sent to coupler

INTERFACE:

```
subroutine diags_cpl_out
```

DESCRIPTION:

Printout grid point info sent to coupler

USES:

```
use ice_flux
use ice_albedo
use ice_calendar
use ice_state
```

INPUT/OUTPUT PARAMETERS:

4.24 Fortran: Module Interface ice_read_write - routines for opening, reading and writing external files (Source File: ice_read_write.F)

Routines for opening, reading and writing external files

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

INTERFACE:

```
module ice_read_write
```

USES:

```
use ice_model_size
use ice_domain
use ice_mpi_internal
```

4.24.1 ice_open - opens an unformatted file for reading

INTERFACE:

```
subroutine ice_open(nu,filename,nbits)
```

DESCRIPTION:

Opens an unformatted file for reading
nbits indicates whether the file is sequential or direct access

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nu,nbits
character (*) :: filename
```

4.24.2 ice_read - reads an unformatted file

INTERFACE:

```
subroutine ice_read(nu,nrec,work,atype,scatter)
```

DESCRIPTION:

Reads an unformatted file
work is a real array, atype indicates the format of the data

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nu,nrec
real (kind=dbl_kind) ::  work(ilo:ihi,jlo:jhi)
character (len=4) atype  ! real,int
logical (kind=log_kind) :: scatter
```

4.24.3 ice_write - writes an unformatted file

INTERFACE:

```
subroutine ice_write(nu,nrec,work,atype,gather)
```

DESCRIPTION:

Writes an unformatted file
work is a real array, atype indicates the format of the data

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: nu,nrec
real (kind=dbl_kind) ::  work(ilo:ihi,jlo:jhi)
character (len=4) atype  ! real,int
logical (kind=log_kind) :: gather
```

4.25 Fortran: Module Interface ice_scaling - scale ice (Source File: ice_scaling.F)

INTERFACE:

```
module ice_scaling
```

REVISION HISTORY:

author: C.M.Bitz ?

USES:

```
use ice_domain
use ice_kinds_mod
use ice_constants
use ice_state
```

4.25.1 scale_fluxes - scale fluxes

INTERFACE:

```
subroutine scale_fluxes
```

REVISION HISTORY:

author: C.M.Bitz ?

USES:

```
use ice_flux
use ice_albedo
use ice_history
use ice_grid
```

INPUT/OUTPUT PARAMETERS:

4.25.2 scale_hist_fluxes - scale history fluxes

INTERFACE:

```
subroutine scale_hist_fluxes
```

REVISION HISTORY:

author: C.M.Bitz ?

USES:

```
use ice_flux
use ice_albedo
use ice_history
use ice_grid
```

INPUT/OUTPUT PARAMETERS:

4.26 Fortran: Module Interface ice_state - primary state variables (Source File: ice_state.F)

Primary state variables in various configurations
Note: other state variables are at the end of this... The primary state variable names are:

main——single-column–single column,—aggregated—units
————————————single category—over cats

aicen(i,j,n)—ain(n)———ai————aice(i,j)———
vicen(i,j,n)—vin(n)———vi or hi———vice(i,j)——m
vsnon(i,j,n)—vsn(n)———vs or hs———vsno(i,j)——m
eicen(i,j,k)—ein(k,n)——-ei(k)————eice(i,j)——J/m2
Tsfcn(i,j,n)—Tsfn(n)——-Tsf————-Tsfc(i,j)——deg

Area is dimensionless because aice is the fractional area
(normalized so that the sum over all categories, including open water, is one). That is why vice/vsno have units of m instead of m3, and eice/esno have units of J/m2 instead of J.

Note that variable names follow 2 simple rules:

(1) If a variable has dimensions i and j, write 'ice' or 'sno' or 'sfc'; else abbreviate as 'i' or 's' or 'sf'.
(2) If a variable has dimension n, add an 'n'; else omit the 'n'.

We will also use the variables hice = vice/aice, hsno = vsno/aice,
qice = eice/vice, and qsno = esno/vsno. These variables will follow the same naming convention. They will not be state variables but will be computed as needed in individual subroutines.

REVISION HISTORY:

```
authors C. M. Bitz, UW
        Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_state
```

USES:

```
use ice_kinds_mod
use ice_model_size
use ice_domain
```

4.27 Fortran: Module Interface ice_therm_driver - driver for thermodynamics and itd (Source File: ice_therm_driver.F)

Energy-conserving sea ice model
Driver for thermodynamics and associated changes to the itd

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669-15,677.

See Bitz, C.M., M.M. Holland, A.J. Weaver, M. Eby, 2001: Simulating the ice-thickness distribution in a climate model, J. Geophys. Res., 106, 2441–2464.

REVISION HISTORY:

```
author C. M. Bitz
code heavily modified by Elizabeth C. Hunke, LANL
```

INTERFACE:

```
module ice_therm_driver
```

USES:

```
use ice_kinds_mod
use ice_model_size
use ice_constants
use ice_state
use ice_flux
use ice_diagnostics
use ice_calendar
```

4.27.1 thermo_rates - compute thermo growth rates and atm fluxes

INTERFACE:

```
subroutine thermo_rates
```

DESCRIPTION:

compute thermodynamic growth rates and atmospheric fluxes

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_domain
use ice_timers
use ice_vthermo
use ice_grid
use ice_itd
use ice_init
```

INPUT/OUTPUT PARAMETERS:

4.27.2 thermo_itd - changes in itd due to thermodynamic growth

INTERFACE:

```
subroutine thermo_itd
```

DESCRIPTION:

Changes to ice thickness distribution associated with thermodynamic growth rates, including lateral growth/melt
NOTE: ocean fluxes are initialized here

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_domain
use ice_timers
use ice_history
use ice_grid
use ice_itd
use ice_itd_linear
use ice_init
```

INPUT/OUTPUT PARAMETERS:

4.27.3 init_column_diags - initialize column diagnostics

INTERFACE:

```
subroutine init_column_diags(i,j,hfatm)
```

DESCRIPTION:

initialize diagnostic and history variables

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_history
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind) :: i, j
real (kind=dbl_kind), intent(out) ::
&   hfatm       ! heat flx  to ice-snow from atm  (J/m**2)
```

4.27.4 init_vertical_profile - initialize vertical ice profile

INTERFACE:

```
subroutine init_vertical_profile(i,j,ei0,hin,hsn,tiz)
```

DESCRIPTION:

initialize the vertical profile of ice properties

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_vthermo
use ice_itd
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i,j

real (kind=dbl_kind), intent(out) ::
&   ei0               ! initial energy in ice/snow        (J/m**2)
&,  hin(ncat)         ! ice thickness for each cat        (m)
&,  hsn(ncat)         ! snow thickness for each cat       (m)
&,  tiz(0:nmax,ncat)  ! temp of each layer for each cat   (C)
```

4.27.5 init_frzmlt - initialize ocean-ice heat fluxes, bottom and lateral

INTERFACE:

```
subroutine init_frzmlt(i,j,Fbot,Fnew,Rside)
```

DESCRIPTION:

Initialize ocean-ice heat fluxes: bottom and lateral
Assuming frzmlt is per grid box area

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_itd
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i, j
real (kind=dbl_kind), intent(out) ::
&   Fbot        ! heat flx to ice bottom,         (W/m**2)
&,  Fnew        ! heat flx to open water,         (W/m**2)
&,  Rside       ! fraction of ice that melts from side
```

4.27.6 lateral_growth_melt - frazil ice growth and lateral melt

INTERFACE:

```
subroutine lateral_growth_melt(i,j,Fnew,Rside,hin)
```

DESCRIPTION:

frazil ice growth / lateral melt

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_vthermo
use ice_history
use ice_itd
```

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: i,j

      real (kind=dbl_kind), intent(in) ::
     &   Fnew        ! heat flx to open water           (W/m**2)
     &,  Rside       ! fraction of ice that melts from side

      real (kind=dbl_kind), intent(inout) ::
     &   hin(ncat)   ! ice thickness for each cat       (m)
```

4.27.7 reduce_area - reduce area when ice melts for special case ncat=1

INTERFACE:

```
      subroutine reduce_area(i,j,dhi1,hin,hsn)
```

DESCRIPTION:

Reduce area when ice melts for special case of ncat=1
Use CSM 1.0-like method of reducing ice area when melting occurs: assume only half the ice volume change goes to thickness decrease, the other half to reduction in ice fraction

REVISION HISTORY:

```
  authors: C.M.Bitz (UW)
  modified by: E.C.Hunke (LANL).
```

USES:

```
      use ice_kinds_mod
      use ice_constants
      use ice_flux  !! not necessary if rebin is called instead
      use ice_state
      use ice_itd
```

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: i,j

      real (kind=dbl_kind), intent(in) ::
     &  dhi1          ! melt at bottom and top plus sublimation  (m)

      real (kind=dbl_kind), intent(inout) ::
     &   hin(ncat)    ! ice thickness for each cat               (m)
     &,  hsn(ncat)    ! ice thickness for each cat               (m)
```

4.27.8 conservation_check - enforce ice property conservation

INTERFACE:

```
      subroutine conservation_check(i,j,ei0,hfatm,hfocn)
```

DESCRIPTION:

Make sure ice properties are properly conserved

REVISION HISTORY:

```
authors: C.M.Bitz (UW)
modified by: E.C.Hunke (LANL).
```

USES:

```
use ice_itd
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i, j

real (kind=dbl_kind), intent(in) ::
&   ei0         ! initial energy in ice/snow      (J/m**2)
&,  hfatm       ! heat flx  to ice-snow from atm  (J/m**2)
&,  hfocn       ! heat flx  to ice-snow from ocn  (J/m**2)
```

4.28 Fortran: Module Interface ice_timers (Source File: ice_timers.F)

Timing routines

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

INTERFACE:

```
module ice_timers
```

DESCRIPTION:

Timing routines

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

```
use ice_kinds_mod
use ice_constants
implicit none
```

4.28.1 ice_timer_clear(n) - initialize timer n to 0

INTERFACE:

```
subroutine ice_timer_clear(n)
```

DESCRIPTION:

Initialize timer n to 0
if n = -1 initialize all timers

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: n
```

4.28.2 ice_timer_start(n) - begin timing with timer n

INTERFACE:

```
subroutine ice_timer_start(n)
```

DESCRIPTION:

Begin timing with timer n

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: n
```

4.28.3 ice_timer_stop(n) - end (or pause) timing with timer n

INTERFACE:

```
subroutine ice_timer_stop(n)
```

DESCRIPTION:

End (or pause) timing with timer n

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: n
```

4.28.4 ice_timer_print(n) - print timing results of timer n

INTERFACE:

```
subroutine ice_timer_print(n)
```

DESCRIPTION:

Print timing results of timer n
if n = -1 print timing results of all timers

REVISION HISTORY:

```
author: Tony Craig, NCAR
```

USES:

```
use ice_domain
use ice_mpi_internal
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: n
```

4.28.5 timers(t1) - do the work

INTERFACE:

```
      subroutine timers(t1)
```

DESCRIPTION:

Do the work

REVISION HISTORY:

```
   author: Tony Craig, NCAR
```

USES:

```
#ifdef _MPI
      include "mpif.h"         ! MPI library definitions
#else
#ifdef Linux
      integer (kind=int_kind) ::
     &     count        !  current count of the system clock
     &,    count_rate   !  number of clock ticks per second
#else
      real (kind=dbl_kind) :: rtc
      integer (kind=int_kind) :: rate,irtc_rate
#endif
#endif
```

INPUT/OUTPUT PARAMETERS:

```
      real (kind=dbl_kind), intent(out) :: t1
```

4.29 Fortran: Module Interface ice_transport - horizontal advection (via mpdata) (Source File: ice_transport.F)

Calculates horizontal advection using mpdata (Multidimensional Positive Definite Advection Transport Algorithm).

REVISION HISTORY:

```
   author Elizabeth C. Hunke
```

INTERFACE:

```
      module ice_transport
```

USES:

```
      use ice_model_size
      use ice_domain
      use ice_constants
      use ice_grid
```

4.29.1 transport - computes transport equations for one timestep

INTERFACE:

```
subroutine transport
```

DESCRIPTION:

Computes the transport equations for one timestep using mpdata. Sets several fields into a work array and passes it to mpdata routine.

REVISION HISTORY:

```
author Elizabeth C. Hunke
```

USES:

```
use ice_flux
use ice_timers
use ice_state
```

INPUT/OUTPUT PARAMETERS:

4.29.2 mpdata(narrays,phi) - advection according to mpdata

INTERFACE:

```
subroutine mpdata(narrays,phi)
```

DESCRIPTION:

Smolarkiewicz, P. K., 1984: A fully multidimensional positive definite advection transport algorithm with small implicit diffusion, J. Comput. Phys., 54, 325-362.

REVISION HISTORY:

```
author Elizabeth C. Hunke
```

USES:

```
use ice_calendar
use ice_dyn_evp
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: narrays

real (kind=dbl_kind), intent(inout) ::
&   phi(imt_local,jmt_local,narrays)
```

4.30 Fortran: Module Interface ice_tstm - energy conserving sea ice model thermodynamics (Source File: ice_tstm.F)

Energy-conserving sea ice model
Routines to solve heat-equation using linear method

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669-15,677.

See Bitz, C.M., M.M. Holland, A.J. Weaver, M. Eby, 2001: Simulating the ice-thickness distribution in a climate model, J. Geophys. Res., 106, 2441–2464.

REVISION HISTORY:

```
author: C. M. Bitz
```

INTERFACE:

```
      module ice_tstm
```

USES:

```
      use ice_constants
      use ice_itd
```

4.30.1 tstm - calculates surface and internal snow/ice temp

INTERFACE:

```
      subroutine tstm(   dtau,  tmz1d,  sal1d,     Tf1
     $                ,  area,     hi,     hs,    dswr
     $                , dswrv,  dswrn,   flwd,  dflwup
     $                ,  dflh,   dfsh,  asnow,      ni
     $                ,  tbot,     Ib,      F,   condb
     $                ,    ts,     ti,  flwup,     flh
     $                ,   fsh)
```

DESCRIPTION:

This routine calculates the evolution of the ice interior and surface temperature from the heat equation and surface energy balance

The albedo is fixed for this calculation

Solves the heat equation which is non-linear for saline ice by linearizing and then iterating a set of equations
Scheme is backwards Euler giving a tridiagonal set of equations implicit in temperature
(Tried crank-nicholson but it behaves poorly when thin ice has a weird initial temperature profile.

NOTE there are two values for minimum snow thickness, hs-min used elsewhere in the code, and hsmin used here. CC says:
The solution to the heat equation ignores the insulating effect of snow if it less than 0.01 m thick, but I do not like to "kill" it when it is that thick because sometimes the snowfall rate is really small...
Must have a sufficient amount of snow to solve heat equation in snow hsmin is the minimum depth of snow in order to solve for ti(0) if snow thickness is less than hsmin then do not change ti(0)

The number of equations that must be solved by the tridiagonal solver depends on whether the surface is melting and whether there is snow. Four cases are possible:

1 = freezing w/ snow, 2 = freezing w/ no snow,
3 = melting w/ snow, and 4 = melting w/ no snow

REVISION HISTORY:

author: C. M. Bitz

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: ni ! number vertical layers in ice
      real (kind=dbl_kind), intent(in) ::
     &   dtau     ! timestep                                      (s)
     &,  tmz1d(ni)! melting temp of each layer                    (C)
     &,  sal1d(ni+1) ! salinity of each layer                     (ppt)
     &,  area     ! area of the ice/snow
     &,  hi,hs    ! ice and snow thickness                        (m)
     &,  asnow    ! 1 - patchy snow frac
     &,  dswr     ! above srfc net dnwd shortwave, positive down (W/m**2)
     &,  dswrv    ! dswr in vis (wvlngth < 700nm)                 (W/m**2)
     &,  dswrn    ! dswr in nir (wvlngth > 700nm)                 (W/m**2)
     &,  flwd     ! dnwd longwave flux (always positive)          (W/m**2)
     &,  dfsh     ! derriv wrt ts of dnwd sensible flux           (W/m**2)
     &,  dflh     ! derriv wrt ts of dnwd latent flux             (W/m**2)
     &,  dflwup   ! derriv wrt ts of upwd longwave flux           (W/m**2)
     &,  Tf1      ! freezing temp of water below ice              (C)

      real (kind=dbl_kind), intent(out) ::
     &   tbot     ! bottom ice temp                               (C)
     &,  condb    ! conductive flux at bottom srfc                (W/m**2)
     &,  F        ! net flux at top srfc including conductive flux (W/m**2)
     &,  Ib       ! solar passing through the bottom ice srfc     (W/m**2)

      real (kind=dbl_kind), intent(inout) ::
     &   ts       ! srfc temp of snow or ice  (C)
     &,  ti(0:ni) ! snow(0) and ice(1:ni) interior temp (C)
     &,  fsh      ! dnwd sensible flux (W/m**2)
     &,  flh      ! dnwd latent flux (always negative) (W/m**2)
     &,  flwup    ! upwd longwave flux (always negative) (W/m**2)
```

4.30.2 getabc - computes elements of tridiagonal matrix

INTERFACE:

```
      subroutine getabc(a,b,c,r,ti,tbot,zeta,k,eta,ni,lfirst)
```

DESCRIPTION:

Compute elements of tridiagonal matrix

REVISION HISTORY:

author: C. M. Bitz

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: ni   ! number of layers
     $       ,lfirst        ! start with this layer

      real (kind=dbl_kind), intent(in) ::
     &        ti   (0:ni)     ! temperature of ice-snow layers
     $       ,tbot            ! temperature of ice bottom srfc
     $       ,zeta (0:nmax)   !
     $       ,k    (0:nmax+1) ! ice-snow conductivitiy
     $       ,eta  (0:nmax)   !

      real (kind=dbl_kind), intent(out) ::
     &        a    (-1:nmax)  ! sub-diagonal elements
     $       ,b    (-1:nmax)  ! diagonal elements
     $       ,c    (-1:nmax)  ! super-diagonal elements
     $       ,r    (-1:nmax)  ! constants (indep. of ti)
```

4.30.3 tridag - solves tridiagonal equations

INTERFACE:

```
      subroutine tridag(a,b,c,r,u,ni)
```

DESCRIPTION:

Solves tridiagonal equations

REVISION HISTORY:

```
   author: C. M. Bitz
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: ni  ! number of rows
      real (kind=dbl_kind), intent(in) ::
     &        a      (nmax)  ! sub-diagonal elements
     $       ,b      (nmax)  ! diagonal elements
     $       ,c      (nmax)  ! super-diagonal elements
     $       ,r      (nmax)  ! constants (indep. of ti)

      real (kind=dbl_kind), intent(out) ::
     &        u      (nmax)  ! solution
```

4.30.4 conductiv - calculates T,S dependent conductivity

INTERFACE:

```
      subroutine conductiv(sal1d,hsmin,ki,ti,tbot,hs,dhi,ni)
```

DESCRIPTION:

Calculates T,S dependent conductivity

REVISION HISTORY:

```
author: C. M. Bitz
```

USES:

INPUT/OUTPUT PARAMETERS:

```
      integer (kind=int_kind), intent(in) :: ni    ! number of layers
      real (kind=dbl_kind), intent(in) ::
     &   hsmin         ! minimum allowable snow thickness for heat eq.
     &,  sal1d(ni+1)   ! salinity of each layer
     &,  hs,ti(0:ni),tbot,dhi
      real (kind=dbl_kind), intent(out) ::
     &   ki(0:nmax+1)
```

4.31 Fortran: Module Interface ice_vthermo - main routines for energy-conserving vertical thermodynamics (Source File: ice_vthermo.F)

Energy-conserving sea ice model
Main routines for vertical thermodynamics

See Bitz, C.M., and W.H. Lipscomb, 1999: An energy-conserving thermodynamic model of sea ice, J. Geophys. Res., 104, 15,669-15,677.

See Bitz, C.M., M.M. Holland, A.J. Weaver, M. Eby, 2001: Simulating the ice-thickness distribution in a climate model, J. Geophys. Res., 106, 2441–2464.

REVISION HISTORY:

```
author C. M. Bitz
```

INTERFACE:

```
      module ice_vthermo
```

USES:

```
      use ice_model_size
      use ice_domain
      use ice_constants
      use ice_state
```

4.31.1 init_thermo - setup salinity profile and tmelt for each layer

INTERFACE:

```
      subroutine init_thermo
```

DESCRIPTION:

Setup the salinity profile and the melting temperature for each layer

REVISION HISTORY:

```
author: C.M.Bitz
```

USES:

```
use ice_itd
use shr_sys_mod, only : shr_sys_abort
```

INPUT/OUTPUT PARAMETERS:

4.31.2 thermo_vertical - heat budget over open water and ice

INTERFACE:

```
subroutine thermo_vertical(i,j,hin,hsn,tiz,Fbot,
&         dhi1,dhin,dhsn,hfatm,hfocn,qin)
```

DESCRIPTION:

Heat budget over open water and ice
NOTE the wind stress is also computed here for later use

REVISION HISTORY:

```
author:  C.M.Bitz
```

USES:

```
use ice_flux
use ice_albedo
use ice_atmo
use ice_history
use ice_calendar
use ice_itd
use ice_dh
use ice_tstm
```

INPUT/OUTPUT PARAMETERS:

```
integer (kind=int_kind), intent(in) :: i, j
real (kind=dbl_kind), intent(in) ::
&   Fbot

real (kind=dbl_kind), intent(inout) ::
&   hin(ncat)          ! ice thickness for each cat              (m)
&,  hsn(ncat)          ! snow thickness for each cat             (m)
&,  tiz(0:nmax,ncat)   ! temp of each layer for each cat         (C)

real (kind=dbl_kind), intent(out) ::
&   dhi1               ! melt at bottom and top plus sublimation  (m)
&,  dhin(ncat)         ! ice thickness change                     (m)
&,  dhsn(ncat)         ! snow thickness change                    (m)
&,  hfatm              ! heat flx  to ice-snow from atm           (J/m**2)
&,  hfocn              ! heat flx  to ice-snow from ocn           (J/m**2)
&,  qin(nmax,ncat)     ! enthalpy per unit volume                 (J/m**3)
```

5 Making Modifications to Code

There are three general types of code modifications one can consider: (1) adding further diagnostics but not changing the bit-for-bit answers, (2) answer changing modifications to the code by way of changing

algorithms, and (3) adding prognostic variables. The first can be as simple as adding new fields to the history file, or producing more diagnostics to the existing code. The second can involve many conceivable changes in the physical algorithms of the sea ice model. The third is the most complex, with the most far-reaching of effects. Each of these will be touched on here, to give the user some indication how to proceed.
Modifications to the sea ice model involve either (or both) modifying existing code and possible additions of new modules. It is strongly recommended that in either case, modifications follow the same general code structure as in the existing code; in particular, using the protex keywords for documentation. New code which adheres to the existing conventions will be more likely to be accepted for inclusion in future releases of CSIM.

5.1 Adding Fields to the History File

The user may add fields to the history file, as follows. The parameter for the number of fields (*avgsiz*) must be increased accordingly (see module *ice_history*), the desired field(s) stored internally in the accumulator array (*aa*), the field name, units and description specified in data statements in the netCDF write routine (*ice_write_hist*), and finally the field(s) included in the appropriate input parameter file. Once these field(s) are added, the standard option to enable/disable history tape fields applies (see the CSIM User's Guide Version 4 for more information).

5.2 Changing Algorithms

The main pitfall in doing this is to be aware of what the coupler and the other CCSM components expect from the ice model. Any changes that effect the coupler exchange, in terms of definitions of fields or even adding/subtracting fields is not recommended. It is best to leave alone the coupler exchange fields when modifying algorithms unless it is crucial for the new algorithm. This would then requre major modifications to the coupler and/or other components, all of which is beyond the scope of this document.

5.3 Adding Prognostic Fields

Adding a new prognostic should not be too difficult. The prognostic can be either category dependent or not. Placing the new field in the module *ice_state* in a manner consistent with the existing prognostics will ensure its availability in existing modules. Careful consideration should be given to where the new prognostic will be updated in the time stepping in the driver file *ice.F* . It might be wise to add the prognostic to the history file for output. It is necessary to add the new prognostic to the restart file if the exact restart character of the sea ice model is to be maintained (crucial for coupled model integrations). It might be wise to test the new prognostic in Active Ice Only (AIO) runs first, before attempting to run fully or partially coupled to other components.

5.4 Modifying Restart Files

Restart file writes and reads can be found in the module *ice_history*. The structure of the restart files can be surmised from the writes and reads in the routines *dumpfile* and *restartfile* respectively. One needs to determine where in the time sequencing the appropriate restart field(s) need(s) to be assigned, and then written by adding extrae write code in *dumpfile*, as well as read code in *restartfile*. Exact restarts can be checked by running a startup run more than two days, rerunning but stopping at least one day short of the first run, and then restarting. The diagnostic log file printout from the sea ice model at the end of the first and the latter two runs should be identical. This ensures bit-for-bit restartability.

6 Miscellaneous Features

The sea ice model has standard diagnostic printout which can be modified. For any such modifications, as well as for those which change the actual simulation, care must be taken not to alter existing budget checks, in particular water and energy. These are briefly discussed here. The diagnostic printout code is found in module *ice_diagnostics*.

6.1 Diagnostic Printouts

Diagnostic output to the log files consists of instantaneous hemispheric mean fields at a user selected frequency. Included is the model step number, the date (yyyymmdd, where yyyy is the year, mm the month, and dd the day), along with the renormalization factors for the conservation of ice after minimum values are removed. Then come a list of fields with max values (max cH) and hemispheric area weighted (arwt) values. The frequency of diagnostic output can be varied: see the CSIM User's Guide Version 4.

6.2 Water and Energy Budgets

Various terms in the water budget are gathered in the routine *runtime_diags* in the module *ice_diagnostics*. Hemispherice (northern and southern) area weighted sums are performed. Modifying the code may require modifying the budget calculation to ensure all terms are included.

7 Summary

This CSIM Code Reference Manual Version 4 gives a brief overview as well as a detailed code listing. It gives suggesions on how to modify the existing code. It should be useful as a starting point and reference for anyone wishing to work with the CSIM source code.

References

Briegleb, B. P., C. M. Bitz, E. C. Hunke, W. H. Lipscomb and J. L. Schramm, 2002: *Description of the Community Climate System Model Version 2 Sea Ice Model*. National Center for Atmospheric Research, http://www.ccsm.ucar.edu/models/ice-csim4.

Schramm, J. L., 2002: *Community Sea Ice Model (CSIM) User's Guide Version 4.0*. National Center for Atmospheric Research, http://www.ccsm.ucar.edu/models/ice-csim4.