Icepack Documentation

CICE Consortium

Apr 26, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION - ICEPACK

1.1 About Icepack

Modern sea ice models have evolved into highly complex collections of physical parameterizations and infrastructural elements to support various configurations and computational approaches. In particular, numerical models may now be implemented for unstructured grids, requiring new approaches for referencing information in neighboring grid cells and communication information across grid elements. However, a large portion of the physics in sea ice models can be described in a vertical column, without reference to neighboring grid cells.

The column physics package of the sea ice model CICE, "Icepack", is maintained by the CICE Consortium. This code includes several options for simulating sea ice thermodynamics, mechanical redistribution (ridging) and associated area and thickness changes. In addition, the model supports a number of tracers, including thickness, enthalpy, ice age, first-year ice area, deformed ice area and volume, melt ponds, and biogeochemistry.

Icepack is implemented in CICE as a git submodule. The purpose of Icepack is to provide the column physics model as a separate library for use in other host models such as CICE. Development and testing of CICE and Icepack may be done together, but the repositories are independent. This document describes the Icepack model. The Icepack code is available from https://github.com/CICE-Consortium/Icepack.

Icepack consists of three independent parts, the column physics code, the icepack driver that supports stand-alone testing of the column physics code, and the icepack scripts that build and test the Icepack model. The column physics is called from a host (driver) model on a gridpoint by gridpoint basis. Each gridpoint is independent and the host model stores and passes the model state and forcing to the column physics.

Major changes with each Icepack release (https://github.com/CICE-Consortium/Icepack/releases) will be detailed with the included release notes. Enhancements and bug fixes made to Icepack since the last numbered release can be found on the Icepack wiki (https://github.com/CICE-Consortium/Icepack/wiki/Icepack-Recent-changes). **Please cite any use of the Icepack code.** More information can be found at *Citing the Icepack code*.

This document uses the following text conventions: Variable names used in the code are `typewritten`. Subroutine names are given in *italic*. File and directory names are in **boldface**. A comprehensive index, including glossary of symbols with many of their values, appears at the end of this guide.

1.2 Quick Start

Download the model from the CICE-Consortium repository,
https://github.com/CICE-Consortium/Icepack

Instructions for working in github with Icepack (and CICE) can be found in the CICE Git and Workflow Guide.

From your main Icepack directory, execute:

```
./icepack.setup -c ~/mycase1 -m testmachine
cd ~/mycase1
./icepack.build
./icepack.submit
```

`testmachine` is a generic machine name included with the icepack scripts. The local machine name will have to be substituted for `testmachine` and there are working ports for several different machines. However, it may be necessary to port the model to a new machine. See *Porting* for more information about how to port. See *Scripts* for more information about how to use the icepack.setup and icepack.submit scripts.

Please cite any use of the Icepack code. More information can be found at *Citing the Icepack code*.

1.3 Acknowledgements

This work has been completed through the CICE Consortium and its members with funding through the

- Department of Energy (Los Alamos National Laboratory)
- Department of Defense (Navy)
- Department of Commerce (National Oceanic and Atmospheric Administration)
- National Science Foundation (the National Center for Atmospheric Research)
- Environment and Climate Change Canada.

Special thanks are due to participants from these institutions and many others who contributed to previous versions of CICE or Icepack.

1.4 Citing the Icepack code

Each individual release has its own Digital Object Identifier (DOI), e.g. Icepack v1.2.2 has DOI 10.5281/zenodo.3888633. All versions of this lineage (e.g. Icepack v1) can be cited by using the DOI 10.5281/zenodo.1213462 (https://zenodo.org/record/1213462). This DOI represents all v1 releases, and will always resolve to the latest one. More information can be found by following the DOI link to zenodo.

If you use Icepack, please cite the version number of the code you are using or modifying.

If using code from the CICE-Consortium repository `main` branch that includes modifications that have not yet been released with a version number, then in addition to the most recent version number, the hash at time of download can be cited, determined by executing the command `git log` in your clone.

A hash can also be cited for your own modifications, once they have been committed to a repository branch.

Please also make the CICE Consortium aware of any publications and model use.

1.5 Copyright

CHAPTER

TWO

SCIENCE GUIDE

2.1 Atmosphere and ocean boundary forcing

Table 1: *External forcing data that are relevant to Icepack*

Variable	Description	External Interactions
z_o	Atmosphere level height	From *atmosphere model* to *sea ice model*
$z_{o,s}$	Atmosphere level height (scalar quantities) (optional)	From *atmosphere model* to *sea ice model*
$\vec{U}_a$	Wind velocity	From *atmosphere model* to *sea ice model*
Q_a	Specific humidity	From *atmosphere model* to *sea ice model*
ρ_a	Air density	From *atmosphere model* to *sea ice model*
Θ_a	Air potential temperature	From *atmosphere model* to *sea ice model*
T_a	Air temperature	From *atmosphere model* to *sea ice model*
$F_{sw\downarrow}$	Incoming shortwave radiation (4 bands)	From *atmosphere model* to *sea ice model*
$F_{L\downarrow}$	Incoming longwave radiation	From *atmosphere model* to *sea ice model*
F_{rain}	Rainfall rate	From *atmosphere model* to *sea ice model*
F_{snow}	Snowfall rate	From *atmosphere model* to *sea ice model*
F_{frzmlt}	Freezing/melting potential	From *ocean model* to *sea ice model*
T_w	Sea surface temperature	From *ocean model* to *sea ice model*
S	Sea surface salinity	From *ocean model* to *sea ice model*
∇H_o	Sea surface slope	From *ocean model* via flux coupler to *sea ice model*
h_1	Thickness of first ocean level (optional)	From *ocean model* to *sea ice model*
$\vec{U}_w$	Surface ocean currents	From *ocean model* to *sea ice model* (available in Icepack driver, not used directly in column physics)
$\vec{\tau}_a$	Wind stress	From *sea ice model* to *atmosphere model*
F_s	Sensible heat flux	From *sea ice model* to *atmosphere model*
F_l	Latent heat flux	From *sea ice model* to *atmosphere model*
$F_{L\uparrow}$	Outgoing longwave radiation	From *sea ice model* to *atmosphere model*
F_{evap}	Evaporated water	From *sea ice model* to *atmosphere model*
α	Surface albedo (4 bands)	From *sea ice model* to *atmosphere model*
T_{sfc}	Surface temperature	From *sea ice model* to *atmosphere model*
$F_{sw\Downarrow}$	Penetrating shortwave radiation	From *sea ice model* to *ocean model*
F_{water}	Fresh water flux	From *sea ice model* to *ocean model*
F_{hocn}	Net heat flux to ocean	From *sea ice model* to *ocean model*
F_{salt}	Salt flux	From *sea ice model* to *ocean model*
$\vec{\tau}_w$	Ice-ocean stress	From *sea ice model* to *ocean model*

continues on next page

Table 1 – continued from previous page

Variable	Description	External Interactions
F_{bio}	Biogeochemical fluxes	From *sea ice model* to *ocean model*
a_i	Ice fraction	From *sea ice model* to both *ocean and atmosphere models*
T_a^{ref}	2m reference temperature (diagnostic)	From *sea ice model* to both *ocean and atmosphere models*
Q_a^{ref}	2m reference humidity (diagnostic)	From *sea ice model* to both *ocean and atmosphere models*
F_{swabs}	Absorbed shortwave (diagnostic)	From *sea ice model* to both *ocean and atmosphere models*
$E(f)$	Wave spectrum as a function of frequency	From *ocean surface wave model* to *sea ice model*

The ice fraction a_i (aice) is the total fractional ice coverage of a grid cell. That is, in each cell,

$$\begin{aligned} a_i &= 0 && \text{if there is no ice} \\ a_i &= 1 && \text{if there is no open water} \\ 0 < a_i &< 1 && \text{if there is both ice and open water,} \end{aligned}$$

where a_i is the sum of fractional ice areas for each category of ice. The ice fraction is used by the flux coupler to merge fluxes from the sea ice model with fluxes from the other earth system components. For example, the penetrating shortwave radiation flux, weighted by a_i, is combined with the net shortwave radiation flux through ice-free leads, weighted by $(1 - a_i)$, to obtain the net shortwave flux into the ocean over the entire grid cell. The CESM flux coupler requires the fluxes to be divided by the total ice area so that the ice and land models are treated identically (land also may occupy less than 100% of an atmospheric grid cell). These fluxes are "per unit ice area" rather than "per unit grid cell area."

In some coupled climate models (for example, recent versions of the U.K. Hadley Centre model) the surface air temperature and fluxes are computed within the atmosphere model and are passed to CICE for use in the column physics. In this case the logical parameter `calc_Tsfc` in *ice_therm_vertical* is set to false. The fields `fsurfn` (the net surface heat flux from the atmosphere), `flatn` (the surface latent heat flux), and `fcondtopn` (the conductive flux at the top surface) for each ice thickness category are copied or derived from the input coupler fluxes and are passed to the thermodynamic driver subroutine, *thermo_vertical*. At the end of the time step, the surface temperature and effective conductivity (i.e., thermal conductivity divided by thickness) of the top ice/snow layer in each category are returned to the atmosphere model via the coupler. Since the ice surface temperature is treated explicitly, the effective conductivity may need to be limited to ensure stability. As a result, accuracy may be significantly reduced, especially for thin ice or snow layers. A more stable and accurate procedure would be to compute the temperature profiles for both the atmosphere and ice, together with the surface fluxes, in a single implicit calculation. This was judged impractical, however, given that the atmosphere and sea ice models generally exist on different grids and/or processor sets.

2.1.1 Atmosphere

The wind velocity, specific humidity, air density and potential temperature at the given level height $z_\circ$ (optionally $z_{\circ,s}$, see below) are used to compute transfer coefficients used in formulas for the surface wind stress and turbulent heat fluxes $\vec{\tau}_a$, F_s, and F_l, as described below. The sensible and latent heat fluxes, F_s and F_l, along with shortwave and longwave radiation, $F_{sw\downarrow}$, $F_{L\downarrow}$ and $F_{L\uparrow}$, are included in the flux balance that determines the ice or snow surface temperature when `calc_Tsfc` is true. As described in the *Thermodynamics* section, these fluxes depend nonlinearly on the ice surface temperature T_{sfc}. The balance equation is iterated until convergence, and the resulting fluxes and T_{sfc} are then passed to the flux coupler.

The snowfall precipitation rate (provided as liquid water equivalent and converted by the ice model to snow depth) also contributes to the heat and water mass budgets of the ice layer. Melt ponds generally form on the ice surface in the Arctic and refreeze later in the fall, reducing the total amount of fresh water that reaches the ocean and altering the heat budget of the ice; this version includes two new melt pond parameterizations. Rain and all melted snow end up in the ocean.

Wind stress and transfer coefficients for the turbulent heat fluxes are computed in subroutine *atmo_boundary_layer* following [32], with additions and changes as detailed in Appendix A of [55] for high frequency coupling (namelist variable `highfreq`). The resulting equations are provided here for the default boundary layer scheme, which is based on Monin-Obukhov theory (`atmbndy = 'stability'`). Alternatively, `atmbndy = 'constant'` provides constant coefficients for wind stress, sensible heat and latent heat calculations (computed in subroutine *atmo_boundary_const*); `atmbndy = 'mixed'` uses the stability based calculation for wind stress and constant coefficients for sensible and latent heat fluxes.

The wind stress and turbulent heat flux calculation accounts for both stable and unstable atmosphere–ice boundary layers. We first define the "stability"

$$\Upsilon = \frac{\kappa g z_\circ}{u^{*2}} \left(\frac{\Theta^*}{\Theta_a (1 + 0.606 Q_a)} + \frac{Q^*}{1/0.606 + Q_a} \right), \tag{2.1}$$

where κ is the von Karman constant, g is gravitational acceleration, and u^*, Θ^* and Q^* are turbulent scales for velocity difference, temperature, and humidity, respectively, computed as (given the ice velocity $\vec{U}_i$):

$$\begin{aligned} u^* = &\quad c_u \max\left(U_{\Delta\text{min}}, \left|\vec{U}_a - \vec{U}_i\right|\right), \\ \Theta^* = &\quad c_\theta \left(\Theta_a - T_{sfc}\right), \\ Q^* = &\quad c_q \left(Q_a - Q_{sfc}\right). \end{aligned} \tag{2.2}$$

Note that *atmo_boundary_layer* also accepts an optional argument, `zlvs`, to support staggered atmospheric levels, i.e. receiving scalar quantities from the atmospheric model (humidity and temperature) at a different vertical level than the winds. In that case a separate stability Υ_s is computed using the same formula as above but substituting z_o by $z_{o,s}$.

Within the u^* expression, $U_{\Delta\text{min}}$ is the minimum allowable value of $|\vec{U}_a - \vec{U}_i|$, which is set to of 0.5 m/s for high frequency coupling (`highfreq` =.true.). When high frequency coupling is turned off (`highfreq` =.false.), it is assumed in equation (2.2) that:

$$\vec{U}_a - \vec{U}_i \approx \vec{U}_a \tag{2.3}$$

and a higher threshold is taken for $U_{\Delta\text{min}}$ of 1m/s. Equation (2.3) is a poor assumption when resolving inertial oscillations in ice-ocean configurations where the ice velocity vector may make a complete rotation over a period of ≥ 11.96 hours, as discussed in [55]. However, (2.3) is acceptable for low frequency ice-ocean coupling on the order of a day or more, when transient ice-ocean Ekman transport is effectively filtered from the model solution. For the Θ^* and Q^* terms in (2.2), T_{sfc} and Q_{sfc} are the surface temperature and specific humidity, respectively. The latter is calculated by assuming a saturated surface, as described in the *Thermodynamic surface forcing balance* section.

Neglecting form drag, the exchange coefficients c_u, c_θ and c_q are initialized as

$$\frac{\kappa}{\ln(z_{ref}/z_{ice})} \tag{2.4}$$

and updated during a short iteration, as they depend upon the turbulent scales. The number of iterations is set by the namelist variable `natmiter`, nominally set to five but sometimes increased by users employing the `highfreq` option. A convergence tolerance `atmiter_conv` on `ustar` can be set to exit the `natmiter` loop early if desired. Here, z_{ref} is a reference height of 10m and z_{ice} is the roughness length scale for the given sea ice category. Υ is constrained to have magnitude less than 10. Further, defining $\chi = (1 - 16\Upsilon)^{0.25}$ and $\chi \geq 1$, the "integrated flux profiles" for momentum and stability in the unstable ($\Upsilon < 0$) case are given by

$$\begin{aligned} \psi_m = &\quad 2\ln\left[0.5(1+\chi)\right] + \ln\left[0.5(1+\chi^2)\right] - 2\tan^{-1}\chi + \frac{\pi}{2}, \\ \psi_s = &\quad 2\ln\left[0.5(1+\chi^2)\right]. \end{aligned} \tag{2.5}$$

In a departure from the parameterization used in [32], we use profiles for the stable case following [31],

$$\psi_m = \psi_s = -\left[0.7\Upsilon + 0.75\left(\Upsilon - 14.3\right)\exp\left(-0.35\Upsilon\right) + 10.7\right]. \tag{2.6}$$

The coefficients are then updated as

$$\begin{aligned} c_u' &= \frac{c_u}{1 + c_u\left(\lambda - \psi_m\right)/\kappa} \\ c_\theta' &= \frac{c_\theta}{1 + c_\theta\left(\lambda_s - \psi_s\right)/\kappa} \\ c_q' &= c_\theta' \end{aligned} \tag{2.7}$$

where $\lambda = \ln\left(z_\circ/z_{ref}\right)$ and $\lambda_s = \ln\left(z_{\circ,s}/z_{ref}\right)$ if staggered atmospheric levels are used, else $\lambda_s = \lambda$. The first iteration ends with new turbulent scales from equations (2.2). After `natmiter` iterations the latent and sensible heat flux coefficients are computed, along with the wind stress:

$$\begin{aligned} C_l &= \rho_a\left(L_{vap} + L_{ice}\right)u^* c_q \\ C_s &= \rho_a c_p u^* c_\theta^* + 1 \\ \vec{\tau}_a &= \frac{\rho_a (u^*)^2\left(\vec{U}_a - \vec{U}_i\right)}{\left|\vec{U}_a - \vec{U}_i\right|} \end{aligned} \tag{2.8}$$

where L_{vap} and L_{ice} are latent heats of vaporization and fusion, ρ_a is the density of air and c_p is its specific heat. Again following [31], we have added a constant to the sensible heat flux coefficient in order to allow some heat to pass between the atmosphere and the ice surface in stable, calm conditions. For the atmospheric stress term in (2.8), we make the assumption in (2.3) when `highfreq` =.false..

The atmospheric reference temperature T_a^{ref} is computed from T_a and T_{sfc} using the coefficients c_u, c_θ and c_q. Although the sea ice model does not use this quantity, it is convenient for the ice model to perform this calculation. The atmospheric reference temperature is returned to the flux coupler as a climate diagnostic. The same is true for the reference humidity, Q_a^{ref}.

Additional details about the latent and sensible heat fluxes and other quantities referred to here can be found in the *Thermodynamic surface forcing balance* section.

2.1.2 Ocean

New sea ice forms when the ocean temperature drops below its freezing temperature. In the Bitz and Lipscomb thermodynamics, [6] $T_f = -\mu S$, where S is the seawater salinity and $\mu = 0.054^\circ$/ppt is the ratio of the freezing temperature of brine to its salinity (linear liquidus approximation). For the mushy thermodynamics, T_f is given by a piecewise linear liquidus relation. The ocean model calculates the new ice formation; if the freezing/melting potential F_{frzmlt} is positive, its value represents a certain amount of frazil ice that has formed in one or more layers of the ocean and floated to the surface. (The ocean model assumes that the amount of new ice implied by the freezing potential actually forms.)

If F_{frzmlt} is negative, it is used to heat already existing ice from below. In particular, the sea surface temperature and salinity are used to compute an oceanic heat flux F_w ($|F_w| \leq |F_{frzmlt}|$) which is applied at the bottom of the ice. The portion of the melting potential actually used to melt ice is returned to the coupler in F_{hocn}. The ocean model adjusts its own heat budget with this quantity, assuming that the rest of the flux remained in the ocean.

In addition to runoff from rain and melted snow, the fresh water flux F_{water} includes ice melt water from the top surface and water frozen (a negative flux) or melted at the bottom surface of the ice. This flux is computed as the net change of fresh water in the ice and snow volume over the coupling time step, excluding frazil ice formation and newly accumulated snow.

Setting the namelist option `update_ocn_f` to true causes frazil ice to be included in the fresh water and salt fluxes. Some ocean models compute the frazil ice fluxes, which then might need to be corrected for consistency with mushy physics. This behavior is controlled using a combination of `update_ocn_f`, `cpl_frazil` and `ktherm`. In particular, `cpl_frazil = 'external'` assumes that the frazil ice fluxes are handled entirely outside of Icepack. When `ktherm=2`, `cpl_frazil = 'fresh_ice_correction'` sends coupling fluxes representing the difference between the mushy frazil fluxes and fluxes computed assuming the frazil is purely fresh ice. Otherwise the internally computed frazil fluxes are sent to the coupler.

There is a flux of salt into the ocean under melting conditions, and a (negative) flux when sea water is freezing. However, melting sea ice ultimately freshens the top ocean layer, since the ocean is much more saline than the ice. The ice model passes the net flux of salt F_{salt} to the flux coupler, based on the net change in salt for ice in all categories. In the present configuration, `ice_ref_salinity` is used for computing the salt flux, although the ice salinity used in the thermodynamic calculation has differing values in the ice layers.

A fraction of the incoming shortwave $F_{sw\Downarrow}$ penetrates the snow and ice layers and passes into the ocean, as described in the *Thermodynamic surface forcing balance* section.

A thermodynamic slab ocean mixed-layer parameterization is available in **icepack_ocean.F90** and can be run in the full CICE configuration. The turbulent fluxes are computed above the water surface using the same parameterizations as for sea ice, but with parameters appropriate for the ocean. The surface flux balance takes into account the turbulent fluxes, oceanic heat fluxes from below the mixed layer, and shortwave and longwave radiation, including that passing through the sea ice into the ocean. If the resulting sea surface temperature falls below the salinity-dependent freezing point, then new ice (frazil) forms. Otherwise, heat is made available for melting the ice.

When the namelist option `calc_dragio` is set to true, the ice-ocean drag coefficient, c_w (`dragio`), is computed from the thickness of the first ocean level, h_1 (`thickness_ocn_layer1`), and an under-ice roughness length, z_{io} (`iceruf_ocn`). The computation follows [58] :

$$c_w = c_w^* \lambda^2 \tag{2.9}$$

where

$$\begin{aligned} c_w^* &= \frac{\kappa^2}{\ln^2\left(h_1/z_{io}\right)}, \\ \lambda &= \frac{h_1 - z_{io}}{h_1\left[\sqrt{c_w^*}\kappa^{-1}\left(\ln(2) - 1 + z_{io}/h_1\right) + 1\right]} \end{aligned} \tag{2.10}$$

2.1.3 Variable exchange coefficients

In the default configuration, atmospheric and oceanic neutral drag coefficients (c_u and c_w) are assumed constant in time and space. These constants are chosen to reflect friction associated with an effective sea ice surface roughness at the ice–atmosphere and ice–ocean interfaces. Sea ice (in both Arctic and Antarctic) contains pressure ridges as well as floe and melt pond edges that act as discrete obstructions to the flow of air or water past the ice, and are a source of form drag. Following [70] and based on recent theoretical developments [41][40], the neutral drag coefficients can now be estimated from properties of the ice cover such as ice concentration, vertical extent and area of the ridges, freeboard and floe draft, and size of floes and melt ponds. The new parameterization allows the drag coefficients to be coupled to the sea ice state and therefore to evolve spatially and temporally. This parameterization is contained in the subroutine *neutral_drag_coeffs* and is accessed by setting `formdrag` = true in the namelist. (Note: see also *Known bugs and other issues*.)

Following [70], consider the general case of fluid flow obstructed by N randomly oriented obstacles of height H and transverse length L_y, distributed on a domain surface area S_T. Under the assumption of a logarithmic fluid velocity profile, the general formulation of the form drag coefficient can be expressed as

$$C_d = \frac{NcS_c^2\gamma L_y H}{2S_T}\left[\frac{\ln(H/z_0)}{\ln(z_{ref}/z_0)}\right]^2, \tag{2.11}$$

where z_0 is a roughness length parameter at the top or bottom surface of the ice, γ is a geometric factor, c is the resistance coefficient of a single obstacle, and S_c is a sheltering function that takes into account the shielding effect of the obstacle,

$$S_c = (1 - \exp(-s_l D/H))^{1/2}, \tag{2.12}$$

with D the distance between two obstacles and s_l an attenuation parameter.

As in the original drag formulation in CICE (*Atmosphere* and *Ocean* sections), c_u and c_w along with the transfer coefficients for sensible heat, c_θ, and latent heat, c_q, are initialized to a situation corresponding to neutral atmosphere–ice and ocean–ice boundary layers. The corresponding neutral exchange coefficients are then replaced by coefficients that explicitly account for form drag, expressed in terms of various contributions as

$$\texttt{Cdn\_atm} = \texttt{Cdn\_atm\_rdg} + \texttt{Cdn\_atm\_floe} + \texttt{Cdn\_atm\_skin} + \texttt{Cdn\_atm\_pond}, \tag{2.13}$$

$$\texttt{Cdn\_ocn} = \texttt{Cdn\_ocn\_rdg} + \texttt{Cdn\_ocn\_floe} + \texttt{Cdn\_ocn\_skin}. \tag{2.14}$$

The contributions to form drag from ridges (and keels underneath the ice), floe edges and melt pond edges can be expressed using the general formulation of equation (2.11) (see [70] for details). Individual terms in equation (2.14) are fully described in [70]. Following [3] the skin drag coefficient is parametrized as

$$\texttt{Cdn\_(atm/ocn)\_skin} = a_i \left(1 - m_{(s/k)} \frac{H_{(s/k)}}{D_{(s/k)}}\right) c_{s(s/k)}, \text{ if } \frac{H_{(s/k)}}{D_{(s/k)}} \ge \frac{1}{m_{(s/k)}}, \tag{2.15}$$

where m_s (m_k) is a sheltering parameter that depends on the average sail (keel) height, H_s (H_k), but is often assumed constant, D_s (D_k) is the average distance between sails (keels), and c_{ss} (c_{sk}) is the unobstructed atmospheric (oceanic) skin drag that would be attained in the absence of sails (keels) and with complete ice coverage, $a_{ice} = 1$.

Calculation of equations (2.11) – (2.15) requires that small-scale geometrical properties of the ice cover be related to average grid cell quantities already computed in the sea ice model. These intermediate quantities are briefly presented here and described in more detail in [70]. The sail height is given by

$$H_s = 2\frac{v_{rdg}}{a_{rdg}}\left(\frac{\alpha \tan\alpha_k R_d + \beta \tan\alpha_s R_h}{\phi_r \tan\alpha_k R_d + \phi_k \tan\alpha_s R_h^2}\right), \tag{2.16}$$

and the distance between sails

$$D_s = 2H_s \frac{a_i}{a_{rdg}}\left(\frac{\alpha}{\tan\alpha_s} + \frac{\beta}{\tan\alpha_k}\frac{R_h}{R_d}\right), \tag{2.17}$$

where $0 < \alpha < 1$ and $0 < \beta < 1$ are weight functions, α_s and α_k are the sail and keel slope, ϕ_s and ϕ_k are constant porosities for the sails and keels, and we assume constant ratios for the average keel depth and sail height ($H_k/H_s = R_h$) and for the average distances between keels and between sails ($D_k/D_s = R_d$). With the assumption of hydrostatic equilibrium, the effective ice plus snow freeboard is $H_f = \bar{h}_i(1 - \rho_i/\rho_w) + \bar{h}_s(1 - \rho_s/\rho_w)$, where ρ_i, ρ_w and ρ_s are respectively the densities of sea ice, water and snow, $\bar{h}_i$ is the mean ice thickness and $\bar{h}_s$ is the mean snow thickness (means taken over the ice covered regions). For the melt pond edge elevation we assume that the melt pond surface is at the same level as the ocean surface surrounding the floes [16][17][18] and use the simplification $H_p = H_f$. Finally to estimate the typical floe size L_A, distance between floes, D_F, and melt pond size, L_P we use the parameterizations of [41] to relate these quantities to the ice and pond concentrations. All of these intermediate quantities are available for output, along with `Cdn_atm`, `Cdn_ocn` and the ratio `Cdn_atm_ratio_n` between the total atmospheric drag and the atmospheric neutral drag coefficient.

We assume that the total neutral drag coefficients are thickness category independent, but through their dependance on the diagnostic variables described above, they vary both spatially and temporally. The total drag coefficients and heat transfer coefficients will also depend on the type of stratification of the atmosphere and the ocean, and we use the parameterization described in the *Atmosphere* section that accounts for both stable and unstable atmosphere–ice boundary layers. In contrast to the neutral drag coefficients the stability effect of the atmospheric boundary layer is calculated separately for each ice thickness category.

The transfer coefficient for oceanic heat flux to the bottom of the ice may be varied based on form drag considerations by setting the namelist variable `fbot_xfer_type` to `Cdn_ocn`; this is recommended when using the form drag parameterization. The default value of the transfer coefficient is 0.006 (`fbot_xfer_type = 'constant'`).

2.2 Ice thickness distribution

The Arctic and Antarctic sea ice packs are mixtures of open water, thin first-year ice, thicker multiyear ice, and thick pressure ridges. The thermodynamic and dynamic properties of the ice pack depend on how much ice lies in each thickness range. Thus the basic problem in sea ice modeling is to describe the evolution of the ice thickness distribution (ITD) in time and space.

The fundamental equation solved by CICE is [68]:

$$\frac{\partial g}{\partial t} = -\nabla \cdot (g\mathbf{u}) - \frac{\partial}{\partial h}(fg) + \psi - L, \tag{2.18}$$

where $\mathbf{u}$ is the horizontal ice velocity, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$, f is the rate of thermodynamic ice growth, ψ is a ridging redistribution function, L is the lateral melt rate and g is the ice thickness distribution function. We define $g(\mathbf{x}, h, t)\, dh$ as the fractional area covered by ice in the thickness range $(h, h + dh)$ at a given time and location. Icepack represents all of the terms in this equation except for the divergence (the first term on the right).

Equation (2.18) is solved by partitioning the ice pack in each grid cell into discrete thickness categories. The number of categories can be set by the user, with a default value $N_C = 5$. (Five categories, plus open water, are generally sufficient to simulate the annual cycles of ice thickness, ice strength, and surface fluxes [5][38].) Each category n has lower thickness bound H_{n-1} and upper bound H_n. The lower bound of the thinnest ice category, H_0, is set to zero. The other boundaries are chosen with greater resolution for small h, since the properties of the ice pack are especially sensitive to the amount of thin ice [42]. The continuous function $g(h)$ is replaced by the discrete variable a_{in}, defined as the fractional area covered by ice in the open water by a_{i0}, giving $\sum_{n=0}^{N_C} a_{in} = 1$ by definition.

Category boundaries are computed in *init_itd* using one of several formulas, summarized in Table *Lower boundary values*. Setting the namelist variable `kcatbound` equal to 0 or 1 gives lower thickness boundaries for any number of thickness categories N_C. Table *Lower boundary values* shows the boundary values for N_C = 5 and linear remapping of the ice thickness distribution. A third option specifies the boundaries based on the World Meteorological Organization classification; the full WMO thickness distribution is used if N_C = 7; if N_C = 5 or 6, some of the thinner categories are combined. The original formula (`kcatbound` = 0) is the default. Category boundaries differ from those shown in Table *Lower boundary values* for the delta-function ITD. Users may substitute their own preferred boundaries in *init_itd*.

Table *Lower boundary values* shows lower boundary values for thickness categories, in meters, for the three distribution options (``*kcatbound*``) and linear remapping (``*kitd*`` = 1). In the WMO case, the distribution used depends on the number of categories used.

Table 2: *Lower boundary values*

distribution	original	round	WMO		
`kcatbound`	0	1	2		
N_C	5	5	5	6	7
categories	lower bound (m)				
1	0.00	0.00	0.00	0.00	0.00
2	0.64	0.60	0.30	0.15	0.10
3	1.39	1.40	0.70	0.30	0.15
4	2.47	2.40	1.20	0.70	0.30
5	4.57	3.60	2.00	1.20	0.70
6				2.00	1.20
7					2.00

2.3 Joint floe size and thickness distribution

Sizes of individual sea ice floes vary over an extremely broad range, from centimeters to hundreds of kilometers. The floe size distribution (FSD) is a probability function that characterizes this variability [56]. An option to include a prognostic sea ice floe size distribution is available and used if `tr_fsd` is set to true. The scheme is based on the theoretical framework described in [24] for a *joint* floe size and thickness distribution (FSTD), and was implemented by [51].

In this theory, individual floes are identified with a size r and area $x(r)$, where $x(r) = 4\alpha r^2$ for $\alpha = 0.66 < \pi/4$ ([56]). The probability distribution $f(r,h)drdh$ is the fraction of grid surface area covered by ice with thickness between h and $h+dh$ and lateral floe size between r and $r+dr$. The FSTD integrates over all floe sizes and ice thicknesses to unity ($\int_r \int_h F(r,h)drdh = 1$); over all floe sizes to the ITD ($\int_r F(r,h)dr = g(h)$); and over all thicknesses to the FSD ($\int_h F(r,h)dh = f(r)$).

For implementation in CICE, the continuous function $f(r,h)drdh$ is replaced with a product of two discrete variables: a_{in} as defined above and $F_{in,k}$. $F_{in,k}$ is the fraction of ice belonging to thickness category n with lateral floe size belonging to floe size class k (denoted `afsdn` in the code). We then have $\sum_{n=0}^{N_C} \sum_{k=0}^{N_f} a_{in} F_{in,k} = 1$ and $\sum_{k=0}^{N_f} F_{in,k} = 1$. $F_{in,k}$ is carried as an area-weighted tracer. The FSD (continuous function $f(r)dr$ or discrete function f_k, denoted `afsd` in the code) is recovered via $\sum_{n=1}^{N_C} a_{in} F_{in,k} = f_k$.

The FSD may be ignored when considering processes that only modify ice thickness (eg. vertical thermodynamics), and the ITD can be ignored when considering processes that only modify floe sizes (eg. wave fracture). For processes that affect both the ITD and the FSD, (eg. lateral melt), both a_{in} and $F_{in,k}$ are evolved.

The FSTD evolves subject to lateral growth, lateral melt, new ice growth, floe welding and wave fracture, as described in [51] and with some modifications described in [53]. The equation for time evolution of the FSTD is ([24]),

$$\frac{\partial f(r,h)}{\partial t} = -\nabla \cdot (f(r,h)\mathbf{v}) + \mathcal{L}_T + \mathcal{L}_M + \mathcal{L}_W,$$

where the terms on the right hand side represent the effects of advection, thermodynamics, mechanical redistribution and wave fracture respectively. Floe sizes do not explicitly appear in the equations of sea ice motion and therefore the FSTD is advected as an area tracer. We also assume that mechanical redistribution of sea ice through ridging does not impact floe sizes. Thus it remains only to compute the thermodynamic and wave fracture tendencies.

Thermodynamic changes to the FSTD are given by

$$\mathcal{L}_T(r,h) = -\nabla_{(r,h)} \cdot (f(r,h)\mathbf{G}) + \frac{2}{r} f(r,h) G_r + \delta(r - r_{\text{min}})\delta(h - h_{\text{min}})\dot{A}_p + \beta_{\text{weld}}.$$

The first two terms on the right-hand side represent growth and melt of existing floes in thickness and lateral size, at a rate $\mathbf{G} = (G_r, G_h)$. The third term represents growth of new ice: new floes are created at a rate $\dot{A}_p$ in the smallest thickness category and a given lateral size category. If wave forcing is provided, the size of newly formed floes is determined via a tensile stress limitation arising from the wave field ([62], [53]); otherwise, all floes are presumed to grow as pancakes in the smallest floe size category resolved. To allow for the joining of individual floes to one another, we represent the welding together of floes in freezing conditions via the fourth term, β_{weld}, using a coagulation equation.

To compute the impact of wave fracture of the FSD, given a local ocean surface wave spectrum is provided, we generate a realization of the sea surface height field, which is uniquely determined by the spectrum up to a phase. In [24] this phase is randomly chosen, and multiple realizations of the resulting surface height field are used to obtain convergent statistics. However this stochastic component would lead to a model that is not bit-for-bit reproducible. Users can choose in the namelist (via `wave_spec_type`) to run the model with the phase set to be constant to obtain bit-for-bit reproducibility (in which case the fracture code is not run to convergence); or to include the random phase (in which case the fracture code is run to convergence, by generating multiple realizations of sea surface height and adding the resulting fractures to a histogram, until successive histograms are the same to within some small error tolerance); or to exclude wave effects completely (not recommended when using the FSD).

We calculate the number and length of fractures that would occur if waves enter a fully ice-covered region defined in one dimension in the direction of propagation, and then apply the outcome proportionally to the ice-covered fraction in each grid cell. Assuming that sea ice flexes with the sea surface height field, strains are computed on this sub-grid-scale

1D domain. If the strain between successive extrema exceeds a critical value new floes are formed with diameters equal to the distance between the extrema.

Note that tendencies in the FSD are computed using adaptive timestepping to ensure that the FSD is bounded by zero and one (see [25]).

Floe size categories are set in *init_fsd_bounds* using an exponential spacing, beginning at 0.5 m with the largest size resolved set by choice of N_f (`nfsd`), the number of floe size categories. Icepack currently supports `nfsd = 1, 12, 16, 24`. Although `nfsd = 1` tracks the same ice floe diameter as is assumed when `tr_fsd=false`, the processes acting on the floes differ. It is assumed that the floe size lies at the midpoint of each floe size category.

If simulations begin without ice (`ice_ic='none'`), the FSD can emerge without initialization. If simulations begin with ice cover, some initial FSD must be prescribed in `init_fsd`. The default (used for `ice_ic='default'`) is a simple relationship determined from point observations by [49], but its basin-wide applicability has not been tested. In Icepack, `ice_ic='default'` is selected for the slab and the full ITD cells.

The history output includes FSD tendency terms for each of the floe-size-modifying processes. Note that the sum of these does not equal the change in the FSD, as the FSD is also modified by changes in the ITD.

2.4 Tracers

Numerous tracers are available with the column physics. Several of these are required (surface temperature and thickness, salinity and enthalpy of ice and snow layers), and many others are options. For instance, there are tracers to track the age of the ice; the area of first-year ice, fractions of ice area and volume that are level, from which the amount of deformed ice can be calculated; pond area, pond volume and volume of ice covering ponds; a prognostic floe size distribution; snow density, grain size, and ice and liquid content; aerosols, water isotopes, and numerous other biogeochemical tracers. Most of these tracers are presented in later sections. Here we describe the ice age tracers and how tracers may depend on other tracers, using the pond tracers as an example.

2.4.1 Ice age

The age of the ice, τ_{age}, is treated as an ice-volume tracer (*trcr_depend* = 1). It is initialized at 0 when ice forms as frazil, and the ice ages the length of the timestep during each timestep. Freezing directly onto the bottom of the ice does not affect the age, nor does melting. Mechanical redistribution processes and advection alter the age of ice in any given grid cell in a conservative manner following changes in ice area. The sea ice age tracer is validated in [26].

Another age-related tracer, the area covered by first-year ice a_{FY}, is an area tracer (*trcr_depend* = 0) that corresponds more closely to satellite-derived ice age data for first-year ice than does τ_{age}. It is re-initialized each year on 15 September (`yday` = 259) in the northern hemisphere and 15 March (`yday` = 75) in the southern hemisphere, in non-leap years. This tracer is increased when new ice forms in open water, in subroutine *add_new_ice* in **icepack_therm_itd.F90**. The first-year area tracer is discussed in [2].

2.4.2 Tracers that depend on other tracers

Tracers may be defined that depend on other tracers. Melt pond tracers provide an example (these equations pertain to topo tracers; level-ice tracers are similar with an extra factor of a_{lvl}, see Equations (2.69)–(2.72)). Conservation equations for pond area fraction $a_{pnd}a_i$ and pond volume $h_{pnd}a_{pnd}a_i$, given the ice velocity $\mathbf{u}$, are

$$\frac{\partial}{\partial t}(a_{pnd}a_i) + \nabla \cdot (a_{pnd}a_i\mathbf{u}) = 0, \tag{2.19}$$

$$\frac{\partial}{\partial t}(h_{pnd}a_{pnd}a_i) + \nabla \cdot (h_{pnd}a_{pnd}a_i\mathbf{u}) = 0. \tag{2.20}$$

These equations represent quantities within one thickness category; all melt pond calculations are performed for each category, separately. Equation (2.20) expresses conservation of melt pond volume, but in this form highlights that the quantity tracked in the code is the pond depth tracer h_{pnd}, which depends on the pond area tracer a_{pnd}. Likewise, a_{pnd} is a tracer on ice area (Equation (2.19)), which is a state variable, not a tracer.

For a generic quantity q that represents a mean value over the ice fraction, qa_i is the average value over the grid cell. Thus for topo melt ponds, h_{pnd} can be considered the actual pond depth, $h_{pnd}a_{pnd}$ is the mean pond depth over the sea ice, and $h_{pnd}a_{pnd}a_i$ is the mean pond depth over the grid cell. These quantities are illustrated in Figure *Melt pond tracer definitions*. The graphic on the right illustrates the *grid cell* fraction of ponds or level ice as defined by the tracers. The chart on the left provides corresponding ice thickness and pond depth averages over the grid cell, sea ice and pond area fractions.

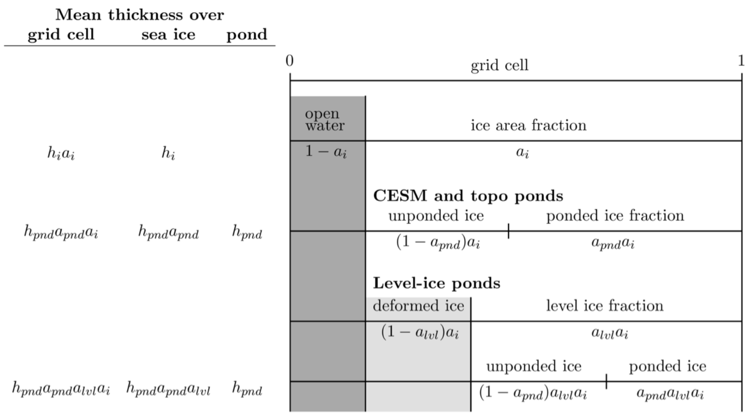

Fig. 1: *Melt pond tracer definitions*

Tracers may need to be modified for physical reasons outside of the "core" module or subroutine describing their evolution. For example, when new ice forms in open water, the new ice does not yet have ponds on it. Likewise when sea ice deforms, we assume that pond water (and ice) on the portion of ice that ridges is lost to the ocean.

When new ice is added to a grid cell, the *grid cell* total area of melt ponds is preserved within each category gaining ice, $a_{pnd}^{t+\Delta t}a_i^{t+\Delta t} = a_{pnd}^t a_i^t$, or

$$a_{pnd}^{t+\Delta t} = \frac{a_{pnd}^t a_i^t}{a_i^{t+\Delta t}}. \tag{2.21}$$

Similar calculations are performed for all tracer types, for example tracer-on-tracer dependencies such as h_{pnd}, when needed:

$$h_{pnd}^{t+\Delta t} = \frac{h_{pnd}^t a_{pnd}^t a_i^t}{a_{pnd}^{t+\Delta t} a_i^{t+\Delta t}}. \tag{2.22}$$

In this case (adding new ice), h_{pnd} does not change because $a_{pnd}^{t+\Delta t}a_i^{t+\Delta t} = a_{pnd}^t a_i^t$.

When ice is transferred between two thickness categories, we conserve the total pond area summed over categories n,

$$\sum_n a_{pnd}^{t+\Delta t}(n) a_i^{t+\Delta t}(n) = \sum_n a_{pnd}^t(n) a_i^t(n). \tag{2.23}$$

Thus,

$$\begin{aligned} a_{pnd}^{t+\Delta t}(m) &= \frac{\sum_n a_{pnd}^t(n) a_i^t(n) - \sum_{n\neq m} a_{pnd}^{t+\Delta t}(n) a_i^{t+\Delta t}(n)}{a_i^{t+\Delta t}(m)} \\ &= \frac{a_{pnd}^t(m) a_i^t(m) + \sum_{n\neq m} \Delta\left(a_{pnd}a_i\right)^{t+\Delta t}}{a_i^{t+\Delta t}(m)} \end{aligned} \tag{2.24}$$

This is more complicated because of the Δ term on the right-hand side, which is handled in subroutine *icepack_compute_tracers*. Such tracer calculations are scattered throughout the code, wherever there are changes to the ice thickness distribution.

Note that if a quantity such as a_{pnd} becomes zero in a grid cell's thickness category, then all tracers that depend on it also become zero. If a tracer should be conserved (e.g., aerosols and the liquid water in topo ponds), additional code must be added to track changes in the conserved quantity.

Tracer dependencies and conserved quantities associated with tracers are tracked using the arrays `trcr_depend`, which defines the type of dependency (area, volume, snow, etc), `n_trcr_strata`, the number of underlying layers, `nt_strata`, the indices of the underlying layers, and `trcr_base`, a mask that is one for the tracer dependency and zero otherwise. These arrays are used to convert between the tracer values themselves and the conserved forms.

More information about the melt pond schemes is in the *Melt ponds* section.

2.5 Transport in thickness space

Next we solve the equation for ice transport in thickness space due to thermodynamic growth and melt,

$$\frac{\partial g}{\partial t} + \frac{\partial}{\partial h}(fg) = 0, \tag{2.25}$$

which is obtained from Equation (2.18) by neglecting the first and third terms on the right-hand side. We use the remapping method of [38], in which thickness categories are represented as Lagrangian grid cells whose boundaries are projected forward in time. The thickness distribution function g is approximated as a linear function of h in each displaced category and is then remapped onto the original thickness categories. This method is numerically smooth and is not too diffusive. It can be viewed as a 1D simplification of the 2D incremental remapping scheme described above.

We first compute the displacement of category boundaries in thickness space. Assume that at time m the ice areas a_n^m and mean ice thicknesses h_n^m are known for each thickness category. (For now we omit the subscript i that distinguishes ice from snow.) We use a thermodynamic model (*Thermodynamics*) to compute the new mean thicknesses h_n^{m+1} at time $m+1$. The time step must be small enough that trajectories do not cross; i.e., $h_n^{m+1} < h_{n+1}^{m+1}$ for each pair of adjacent categories. The growth rate at $h = h_n$ is given by $f_n = (h_n^{m+1} - h_n^m)/\Delta t$. By linear interpolation we estimate the growth rate F_n at the upper category boundary H_n:

$$F_n = f_n + \frac{f_{n+1} - f_n}{h_{n+1} - h_n}(H_n - h_n). \tag{2.26}$$

If a_n or $a_{n+1} = 0$, F_n is set to the growth rate in the nonzero category, and if $a_n = a_{n+1} = 0$, we set $F_n = 0$. The temporary displaced boundaries are given by

$$H_n^* = H_n + F_n\,\Delta t,\ n = 1 \text{ to } N-1 \tag{2.27}$$

The boundaries must not be displaced by more than one category to the left or right; that is, we require $H_{n-1} < H_n^* < H_{n+1}$. Without this requirement we would need to do a general remapping rather than an incremental remapping, at the cost of added complexity.

Next we construct $g(h)$ in the displaced thickness categories. The ice areas in the displaced categories are $a_n^{m+1} = a_n^m$, since area is conserved following the motion in thickness space (i.e., during vertical ice growth or melting). The new ice volumes are $v_n^{m+1} = (a_n h_n)^{m+1} = a_n^m h_n^{m+1}$. For conciseness, define $H_L = H_{n-1}^*$ and $H_R = H_n^*$ and drop the time index $m+1$. We wish to construct a continuous function $g(h)$ within each category such that the total area and volume at time $m+1$ are a_n and v_n, respectively:

$$\int_{H_L}^{H_R} g\,dh = a_n, \tag{2.28}$$

$$\int_{H_L}^{H_R} h\, g\, dh = v_n. \tag{2.29}$$

The simplest polynomial that can satisfy both equations is a line. It is convenient to change coordinates, writing $g(\eta) = g_1\eta + g_0$, where $\eta = h - H_L$ and the coefficients g_0 and g_1 are to be determined. Then Equations (2.28) and (2.29) can be written as

$$g_1 \frac{\eta_R^2}{2} + g_0 \eta_R = a_n, \tag{2.30}$$

$$g_1 \frac{\eta_R^3}{3} + g_0 \frac{\eta_R^2}{2} = a_n \eta_n, \tag{2.31}$$

where $\eta_R = H_R - H_L$ and $\eta_n = h_n - H_L$. These equations have the solution

$$g_0 = \frac{6a_n}{\eta_R^2}\left(\frac{2\eta_R}{3} - \eta_n\right), \tag{2.32}$$

$$g_1 = \frac{12a_n}{\eta_R^3}\left(\eta_n - \frac{\eta_R}{2}\right). \tag{2.33}$$

Since g is linear, its maximum and minimum values lie at the boundaries, $\eta = 0$ and η_R:

$$g(0) = \frac{6a_n}{\eta_R^2}\left(\frac{2\eta_R}{3} - \eta_n\right) = g_0, \tag{2.34}$$

$$g(\eta_R) = \frac{6a_n}{\eta_R^2}\left(\eta_n - \frac{\eta_R}{3}\right). \tag{2.35}$$

Equation (2.34) implies that $g(0) < 0$ when $\eta_n > 2\eta_R/3$, i.e., when h_n lies in the right third of the thickness range (H_L, H_R). Similarly, Equation (2.35) implies that $g(\eta_R) < 0$ when $\eta_n < \eta_R/3$, i.e., when h_n is in the left third of the range. Since negative values of g are unphysical, a different solution is needed when h_n lies outside the central third of the thickness range. If h_n is in the left third of the range, we define a cutoff thickness, $H_C = 3h_n - 2H_L$, and set $g = 0$ between H_C and H_R. Equations (2.32) and (2.30) are then valid with η_R redefined as $H_C - H_L$. And if h_n is in the right third of the range, we define $H_C = 3h_n - 2H_R$ and set $g = 0$ between H_L and H_C. In this case, (2.32) and (2.30) apply with $\eta_R = H_R - H_C$ and $\eta_n = h_n - H_C$.

Figure *Linear approximation of thickness distribution function* illustrates the linear reconstruction of g for the simple cases $H_L = 0$, $H_R = 1$, $a_n = 1$, and $h_n = 0.2$, 0.4, 0.6, and 0.8. Note that g slopes downward ($g_1 < 0$) when h_n is less than the midpoint thickness, $(H_L + H_R)/2 = 1/2$, and upward when h_n exceeds the midpoint thickness. For $h_n = 0.2$ and 0.8, $g = 0$ over part of the range.

Finally, we remap the thickness distribution to the original boundaries by transferring area and volume between categories. We compute the ice area Δa_n and volume Δv_n between each original boundary H_n and displaced boundary H_n^*. If $H_n^* > H_n$, ice moves from category n to $n+1$. The area and volume transferred are

$$\Delta a_n = \int_{H_n}^{H_n^*} g\, dh, \tag{2.36}$$

$$\Delta v_n = \int_{H_n}^{H_n^*} h\, g\, dh. \tag{2.37}$$

If $H_n^* < H_N$, ice area and volume are transferred from category $n+1$ to n using Equations (2.36) and (2.37) with the limits of integration reversed. To evaluate the integrals we change coordinates from h to $\eta = h - H_L$, where H_L is the left limit of the range over which $g > 0$, and write $g(\eta)$ using Equations (2.32) and (2.30). In this way we obtain the new areas a_n and volumes v_n between the original boundaries H_{n-1} and H_n in each category. The new thicknesses, $h_n = v_n/a_n$, are guaranteed to lie in the range (H_{n-1}, H_n). If $g = 0$ in the part of a category that is remapped to a neighboring category, no ice is transferred.

Other conserved quantities are transferred in proportion to the ice volume Δv_{in}. For example, the transferred ice energy in layer k is $\Delta e_{ink} = e_{ink}(\Delta v_{in}/v_{in})$.

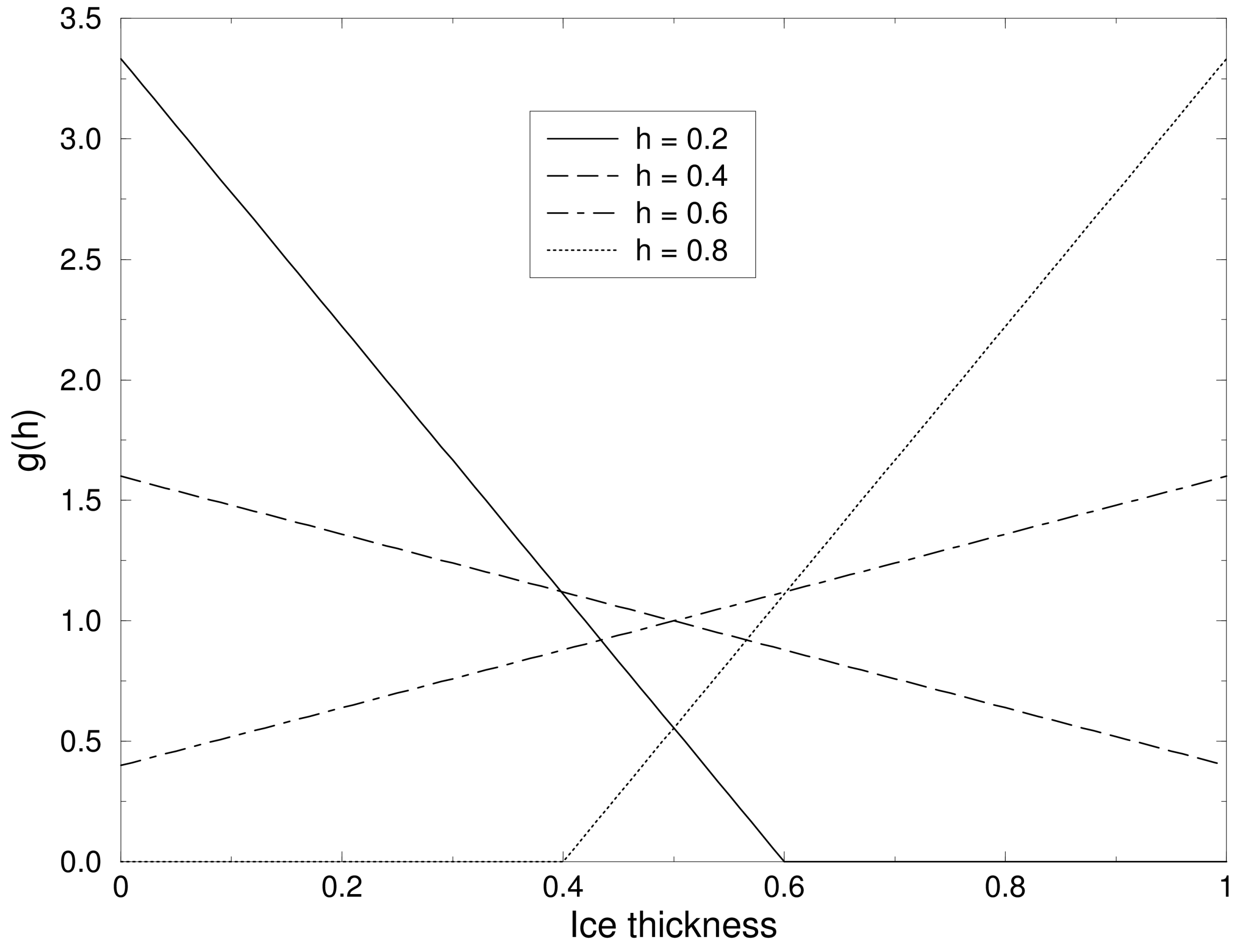

Fig. 2: *Linear approximation of thickness distribution function*

The left and right boundaries of the domain require special treatment. If ice is growing in open water at a rate F_0, the left boundary H_0 is shifted to the right by $F_0 \Delta t$ before g is constructed in category 1, then reset to zero after the remapping is complete. New ice is then added to the grid cell, conserving area, volume, and energy. If ice cannot grow in open water (because the ocean is too warm or the net surface energy flux is downward), H_0 is fixed at zero, and the growth rate at the left boundary is estimated as $F_0 = f_1$. If $F_0 < 0$, all ice thinner than $\Delta h_0 = -F_0 \Delta t$ is assumed to have melted, and the ice area in category 1 is reduced accordingly. The area of new open water is

$$\Delta a_0 = \int_0^{\Delta h_0} g \, dh. \tag{2.38}$$

The right boundary H_N is not fixed but varies with h_N, the mean ice thickness in the thickest category. Given h_N, we set $H_N = 3h_N - 2H_{N-1}$, which ensures that $g(h) > 0$ for $H_{N-1} < h < H_N$ and $g(h) = 0$ for $h \geq H_N$. No ice crosses the right boundary. If the ice growth or melt rates in a given grid cell are too large, the thickness remapping scheme will not work. Instead, the thickness categories in that grid cell are treated as delta functions following [5], and categories outside their prescribed boundaries are merged with neighboring categories as needed. For time steps of less than a day and category thickness ranges of 10 cm or more, this simplification is needed rarely, if ever.

The linear remapping algorithm for thickness is not monotonic for tracers, although significant errors rarely occur. Usually they appear as snow temperatures (enthalpy) outside the physical range of values in very small snow volumes. In this case we transfer the snow and its heat and tracer contents to the ocean.

2.6 Mechanical redistribution

The last term on the right-hand side of Equation (2.18) is ψ, which describes the redistribution of ice in thickness space due to ridging and other mechanical processes. The mechanical redistribution scheme in Icepack is based on [68], [57], [22], [15], and [39]. This scheme converts thinner ice to thicker ice and is applied after horizontal transport. When the ice is converging, enough ice ridges to ensure that the ice area does not exceed the grid cell area.

First we specify the participation function: the thickness distribution $a_P(h) = b(h)\, g(h)$ of the ice participating in ridging. (We use "ridging" as shorthand for all forms of mechanical redistribution, including rafting.) The weighting function $b(h)$ favors ridging of thin ice and closing of open water in preference to ridging of thicker ice. There are two options for the form of $b(h)$. If `krdg_partic = 0` in the namelist, we follow [68] and set

$$b(h) = \begin{cases} \frac{2}{G^*}\left(1 - \frac{G(h)}{G^*}\right) & \text{if } G(h) < G^* \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

where $G(h)$ is the fractional area covered by ice thinner than h, and G^* is an empirical constant. Integrating $a_P(h)$ between category boundaries H_{n-1} and H_n, we obtain the mean value of a_P in category n:

$$a_{Pn} = \frac{2}{G^*}(G_n - G_{n-1}) \left(1 - \frac{G_{n-1} + G_n}{2G^*}\right), \tag{2.40}$$

where a_{Pn} is the ratio of the ice area ridging (or open water area closing) in category n to the total area ridging and closing, and G_n is the total fractional ice area in categories 0 to n. Equation (2.40) applies to categories with $G_n < G^*$. If $G_{n-1} < G^* < G_n$, then Equation (2.40) is valid with G^* replacing G_n, and if $G_{n-1} > G^*$, then $a_{Pn} = 0$. If the open water fraction $a_0 > G^*$, no ice can ridge, because "ridging" simply reduces the area of open water. As in [68] we set $G^* = 0.15$.

If the spatial resolution is too fine for a given time step Δt, the weighting function Equation (2.39) can promote numerical instability. For $\Delta t = 1$ hour, resolutions finer than $\Delta x \sim 10$ km are typically unstable. The instability results from feedback between the ridging scheme and the dynamics via the ice strength. If the strength changes significantly on time scales less than Δt, the viscous-plastic solution of the momentum equation is inaccurate and sometimes oscillatory. As a result, the fields of ice area, thickness, velocity, strength, divergence, and shear can become noisy and unphysical.

A more stable weighting function was suggested by [39]:

$$b(h) = \frac{\exp[-G(h)/a^*]}{a^*[1-\exp(-1/a^*)]} \tag{2.41}$$

When integrated between category boundaries, Equation (2.41) implies

$$a_{Pn} = \frac{\exp(-G_{n-1}/a^*) - \exp(-G_n/a^*)}{1-\exp(-1/a^*)} \tag{2.42}$$

This weighting function is used if `krdg_partic` = 1 in the namelist. From Equation (2.41), the mean value of G for ice participating in ridging is a^*, as compared to $G^*/3$ for Equation (2.39). For typical ice thickness distributions, setting $a^* = 0.05$ with `krdg_partic` = 1 gives participation fractions similar to those given by $G^* = 0.15$ with `krdg_partic` = 0. See [39] for a detailed comparison of these two participation functions.

Thin ice is converted to thick, ridged ice in a way that reduces the total ice area while conserving ice volume and internal energy. There are two namelist options for redistributing ice among thickness categories. If `krdg_redist` = 0, ridging ice of thickness h_n forms ridges whose area is distributed uniformly between $H_{\mathrm{min}} = 2h_n$ and $H_{\mathrm{max}} = 2\sqrt{H^* h_n}$, as in [22]. The default value of H^* is 25 m, as in earlier versions of CICE. Observations suggest that $H^* = 50$ m gives a better fit to first-year ridges [1], although the lower value may be appropriate for multiyear ridges [15]. The ratio of the mean ridge thickness to the thickness of ridging ice is $k_n = (H_{\mathrm{min}} + H_{\mathrm{max}})/(2h_n)$. If the area of category n is reduced by ridging at the rate r_n, the area of thicker categories grows simultaneously at the rate r_n/k_n. Thus the *net* rate of area loss due to ridging of ice in category n is $r_n(1-1/k_n)$.

The ridged ice area and volume are apportioned among categories in the thickness range $(H_{\mathrm{min}}, H_{\mathrm{max}})$. The fraction of the new ridge area in category m is

$$f_m^{\mathrm{area}} = \frac{H_R - H_L}{H_{\mathrm{max}} - H_{\mathrm{min}}}, \tag{2.43}$$

where $H_L = \max(H_{m-1}, H_{\mathrm{min}})$ and $H_R = \min(H_m, H_{\mathrm{max}})$. The fraction of the ridge volume going to category m is

$$f_m^{\mathrm{vol}} = \frac{(H_R)^2 - (H_L)^2}{(H_{\mathrm{max}})^2 - (H_{\mathrm{min}})^2}. \tag{2.44}$$

This uniform redistribution function tends to produce too little ice in the 3–5 m range and too much ice thicker than 10 m [1]. Observations show that the ITD of ridges is better approximated by a negative exponential. Setting `krdg_redist` = 1 gives ridges with an exponential ITD [39]:

$$g_R(h) \propto \exp[-(h - H_{\mathrm{min}})/\lambda] \tag{2.45}$$

for $h \geq H_{\mathrm{min}}$, with $g_R(h) = 0$ for $h < H_{\mathrm{min}}$. Here, λ is an empirical *e*-folding scale and $H_{\mathrm{min}} = 2h_n$ (where h_n is the thickness of ridging ice). We assume that $\lambda = \mu h_n^{1/2}$, where μ (mu_rdg) is a tunable parameter with units . Thus the mean ridge thickness increases in proportion to $h_n^{1/2}$, as in [22]. The value $\mu = 4.0$ gives λ in the range 1–4 m for most ridged ice. Ice strengths with $\mu = 4.0$ and `krdg_redist` = 1 are roughly comparable to the strengths with $H^* = 50$ m and `krdg_redist` = 0.

From Equation (2.45) it can be shown that the fractional area going to category m as a result of ridging is

$$f_m^{\mathrm{area}} = \exp[-(H_{m-1} - H_{\mathrm{min}})/\lambda] - \exp[-(H_m - H_{\mathrm{min}})/\lambda]. \tag{2.46}$$

The fractional volume going to category m is

$$f_m^{\mathrm{vol}} = \frac{(H_{m-1}+\lambda)\exp[-(H_{m-1} - H_{\mathrm{min}})/\lambda] - (H_m+\lambda)\exp[-(H_m - H_{\mathrm{min}})/\lambda]}{H_{min}+\lambda}. \tag{2.47}$$

Equations (2.46) and (2.47) replace Equations (2.43) and (2.44) when `krdg_redist` = 1.

Internal ice energy is transferred between categories in proportion to ice volume. Snow volume and internal energy are transferred in the same way, except that a fraction of the snow may be deposited in the ocean instead of added to the new ridge.

The net area removed by ridging and closing is a function of the strain rates. Let $R_{\rm net}$ be the net rate of area loss for the ice pack (i.e., the rate of open water area closing, plus the net rate of ice area loss due to ridging). Following [15], $R_{\rm net}$ is given by

$$R_{\rm net} = \frac{C_s}{2}(\Delta - |D_D|) - \min(D_D, 0), \tag{2.48}$$

where C_s is the fraction of shear dissipation energy that contributes to ridge-building, D_D is the divergence, and Δ is a function of the divergence and shear. These strain rates are computed by the dynamics scheme. The default value of C_s is 0.25.

Next, define $R_{\rm tot} = \sum_{n=0}^{N} r_n$. This rate is related to $R_{\rm net}$ by

$$R_{\rm net} = \left[a_{P0} + \sum_{n=1}^{N} a_{Pn} \left(1 - \frac{1}{k_n} \right) \right] R_{\rm tot}. \tag{2.49}$$

Given $R_{\rm net}$ from Equation (2.48), we use Equation (2.49) to compute $R_{\rm tot}$. Then the area ridged in category n is given by $a_{rn} = r_n \Delta t$, where $r_n = a_{Pn} R_{\rm tot}$. The area of new ridges is a_{rn}/k_n, and the volume of new ridges is $a_{rn} h_n$ (since volume is conserved during ridging). We remove the ridging ice from category n and use Equations (2.43) and (2.44) (or (2.46) and (2.47)) to redistribute the ice among thicker categories.

Occasionally the ridging rate in thickness category n may be large enough to ridge the entire area a_n during a time interval less than Δt. In this case $R_{\rm tot}$ is reduced to the value that exactly ridges an area a_n during Δt. After each ridging iteration, the total fractional ice area a_i is computed. If $a_i > 1$, the ridging is repeated with a value of $R_{\rm net}$ sufficient to yield $a_i = 1$.

Two tracers for tracking the ridged ice area and volume are available. The actual tracers are for level (undeformed) ice area (*alvl*) and volume (*vlvl*), which are easier to implement for a couple of reasons: (1) ice ridged in a given thickness category is spread out among the rest of the categories, making it more difficult (and expensive) to track than the level ice remaining behind in the original category; (2) previously ridged ice may ridge again, so that simply adding a volume of freshly ridged ice to the volume of previously ridged ice in a grid cell may be inappropriate. Although the code currently only tracks level ice internally, both level ice and ridged ice are available for output. They are simply related:

$$\begin{aligned} a_{lvl} + a_{rdg} = & \quad a_i, \\ v_{lvl} + v_{rdg} = & \quad v_i. \end{aligned} \tag{2.50}$$

Level ice area fraction and volume increase with new ice formation and decrease steadily via ridging processes. Without the formation of new ice, level ice asymptotes to zero because we assume that both level ice and ridged ice ridge, in proportion to their fractional areas in a grid cell (in the spirit of the ridging calculation itself which does not prefer level ice over previously ridged ice).

The ice strength P may be computed in either of two ways. If the namelist parameter `kstrength` = 0, we use the strength formula from [21]:

$$P = P^* h \exp[-C(1 - a_i)], \tag{2.51}$$

where $P^* = 27,500\,\mathrm{N/m^2}$ and $C = 20$ are empirical constants, and h is the mean ice thickness. Alternatively, setting `kstrength` = 1 gives an ice strength closely related to the ridging scheme. Following [57], the strength is assumed proportional to the change in ice potential energy ΔE_P per unit area of compressive deformation. Given uniform ridge ITDs (`krdg_redist` = 0), we have

$$P = C_f\, C_p\, \beta \sum_{n=1}^{N_C} \left[-a_{Pn}\, h_n^2 + \frac{a_{Pn}}{k_n} \left(\frac{(H_n^{\max})^3 - (H_n^{\min})^3}{3(H_n^{\max} - H_n^{\min})} \right) \right], \tag{2.52}$$

where $C_P = (g/2)(\rho_i/\rho_w)(\rho_w - \rho_i)$, $\beta = R_{\mathrm{tot}}/R_{\mathrm{net}} > 1$ from Equation (2.49), and C_f is an empirical parameter that accounts for frictional energy dissipation. Following [15], we set $C_f = 17$. The first term in the summation is the potential energy of ridging ice, and the second, larger term is the potential energy of the resulting ridges. The factor of β is included because a_{Pn} is normalized with respect to the total area of ice ridging, not the net area removed. Recall that more than one unit area of ice must be ridged to reduce the net ice area by one unit. For exponential ridge ITDs (`krdg_redist` = 1), the ridge potential energy is modified:

$$P = C_f\, C_p\, \beta \sum_{n=1}^{N_C} \left[-a_{Pn}\, h_n^2 + \frac{a_{Pn}}{k_n} \left(H_{\mathrm{min}}^2 + 2H_{\mathrm{min}}\lambda + 2\lambda^2 \right) \right] \tag{2.53}$$

The energy-based ice strength given by Equations (2.52) or (2.53) is more physically realistic than the strength given by Equation (2.51). However, use of Equation (2.51) is less likely to allow numerical instability at a given resolution and time step. See [39] for more details.

2.7 Thermodynamics

The current Icepack version includes two thermodynamics options, the Bitz and Lipscomb model [6] (`ktherm` = 1) that assumes a fixed salinity profile, and a "mushy" formulation (`ktherm` = 2) in which salinity evolves [71]. For each thickness category, Icepack computes changes in the ice and snow thickness and vertical temperature profile resulting from radiative, turbulent, and conductive heat fluxes. The ice has a temperature-dependent specific heat to simulate the effect of brine pocket melting and freezing, for `ktherm` = 1 and 2.

Each thickness category n in each grid cell is treated as a horizontally uniform column with ice thickness $h_{in} = v_{in}/a_{in}$ and snow thickness $h_{sn} = v_{sn}/a_{in}$. (Henceforth we omit the category index n.) Each column is divided into N_i ice layers of thickness $\Delta h_i = h_i/N_i$ and N_s snow layers of thickness $\Delta h_s = h_s/N_s$. Minimum ice and snow thickness is specified by namelist parameters `hi_min` and `hs_min`.

The surface temperature (i.e., the temperature of ice or snow at the interface with the atmosphere) is T_{sf}, which cannot exceed $0°C$. The temperature at the midpoint of the snow layer is T_s, and the midpoint ice layer temperatures are T_{ik}, where k ranges from 1 to N_i. The temperature at the bottom of the ice is held at T_f, the freezing temperature of the ocean mixed layer. All temperatures are in degrees Celsius unless stated otherwise.

The `tfrz_option` namelist specifies the freezing temperature formulation. `minus1p8` fixes the freezing temperature at -1.8C. `constant` fixes the freeing point at whatever value is specified by the parameter `Tocnfrz`. `linear_salt` sets the freezing temperature based on salinity, $Tf = -depressT * sss$. And `mushy` uses the mushy formulation for setting the freezing temperature.

Each ice layer has an enthalpy q_{ik}, defined as the negative of the energy required to melt a unit volume of ice and raise its temperature to $0°C$. Because of internal melting and freezing in brine pockets, the ice enthalpy depends on the brine pocket volume and is a function of temperature and salinity. We can also define a snow enthalpy q_s, which depends on temperature alone.

Given surface forcing at the atmosphere–ice and ice–ocean interfaces along with the ice and snow thicknesses and temperatures/enthalpies at time m, the thermodynamic model advances these quantities to time $m + 1$ (`ktherm` = 2 also advances salinity). The calculation proceeds in two steps. First we solve a set of equations for the new temperatures, as discussed in the *New temperatures* section. Then we compute the melting, if any, of ice or snow at the top surface, and the growth or melting of ice at the bottom surface, as described in the *Growth and melting* section. We begin by describing the melt ponds and surface forcing parameterizations, which are closely related to the ice and snow surface temperatures.

Sometimes instabilities can arise when the temperature is close to the melt point in the snow and sea ice and there is abundant internal shorwave absorbed. One can choose to "move" the excess internal shortwave in this case up to the top surface to be reabsorbed. The namelist parameters for this option are `sw_redist`, `sw_frac`, and `sw_dtemp`. By default, `sw_redist` is set to `.false.`

2.7.1 Snow fraction

In several places in the code, the snow fraction over ice (either sea ice or pond lids) varies as a function of snow depth. That is, thin layers of snow are assumed to be patchy, which allows the shortwave flux to increase gradually as the layer thins, preventing sudden changes in the shortwave reaching the sea ice (which can cause the thermodynamics solver to not converge). For example, the parameter `snowpatch` is used for the CCSM3 radiation scheme, with a default value of 0.02:

$$f_{snow} = \frac{h_s}{h_s + h_{snowpatch}},$$

The parameters `hs0` and `hs1` are used similarly for delta-Eddington radiation calculations with meltponds, with `hs0` over sea ice and `hs1` over pond ice.

In the tests shown in [27], $h_{s0} = 0$ for all cases except with the cesm pond scheme; that pond scheme has now been deprecated. h_{s0} can be used with the topo pond scheme, although its impacts have not been documented. We enforce $hs0 = 0$ for level-ice ponds because the infiltration of snow by pond water accomplishes the gradual radiative forcing transition for which the patchy-snow parameters were originally intended. When level-ice ponds are not used, then a typical value for hs0 is 0.03.

With level-ice ponds, the pond water is allowed to infiltrate snow over the level ice area, invisible to the radiation scheme, until the water becomes deep enough to show through the snow layer. The pond fraction is computed during this process and then used to set the snow fraction such that $f_{snow} + f_{pond} = 1$. The ponds are only on the level ice area, and so there is still snow on the ridges even if the entire level ice area becomes filled with ponds.

See [27] for a discussion of the impacts of varying hs1, whose default value is 0.03.

2.7.2 Melt ponds

Three explicit melt pond parameterizations are available in Icepack, and all must use the delta-Eddington radiation scheme, described below. The ccsm3 shortwave parameterization incorporates melt ponds implicitly by adjusting the albedo based on surface conditions.

For each of the three explicit parameterizations, a volume ΔV_{melt} of melt water produced on a given category may be added to the melt pond liquid volume:

$$\Delta V_{melt} = \frac{r}{\rho_w} \left(\rho_i \Delta h_i + \rho_s \Delta h_s + F_{rain} \Delta t\right) a_i, \tag{2.54}$$

where

$$r = r_{min} + (r_{max} - r_{min})\, a_i \tag{2.55}$$

is the fraction of the total melt water available that is added to the ponds, ρ_i and ρ_s are ice and snow densities, Δh_i and Δh_s are the thicknesses of ice and snow that melted, and F_{rain} is the rainfall rate. Namelist parameters are set for the level-ice (`tr_pond_lvl`) parameterization; in the cesm and topo pond schemes the standard values of r_{max} and r_{min} are 0.7 and 0.15, respectively.

Radiatively, the surface of an ice category is divided into fractions of snow, pond and bare ice. In these melt pond schemes, the actual pond area and depth are maintained throughout the simulation according to the physical processes acting on it. However, snow on the sea ice and pond ice may shield the pond and ice below from solar radiation. These processes do not alter the actual pond volume; instead they are used to define an "effective pond fraction" (and likewise, effective pond depth, snow fraction and snow depth) used only for the shortwave radiation calculation.

In addition to the physical processes discussed below, tracer equations and definitions for melt ponds are also described in the *Tracers* section.

Topographic formulation (`tr_pond_topo` = true)

The principle concept of this scheme is that melt water runs downhill under the influence of gravity and collects on sea ice with increasing surface height starting at the lowest height [16][17][18]. Thus, the topography of the ice cover plays a crucial role in determining the melt pond cover. However, Icepack does not explicitly represent the topography of sea ice. Therefore, we split the existing ice thickness distribution function into a surface height and basal depth distribution assuming that each sea ice thickness category is in hydrostatic equilibrium at the beginning of the melt season. We then calculate the position of sea level assuming that the ice in the whole grid cell is rigid and in hydrostatic equilibrium.

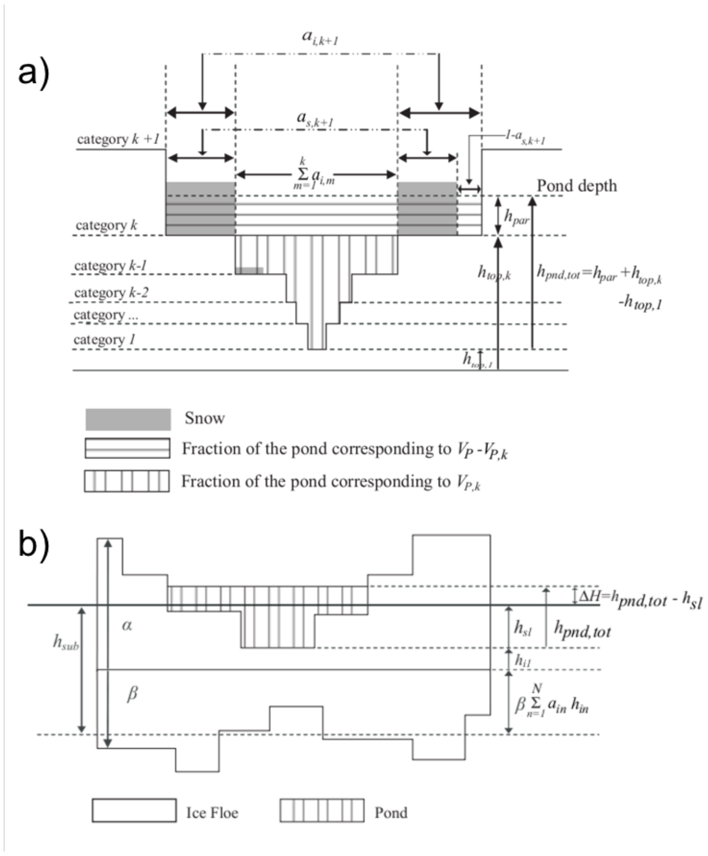

Fig. 3: *Melt Ponds*

Figure *Melt Ponds* illustrates (a) Schematic illustration of the relationship between the height of the pond surface $h_{pnd,tot}$, the volume of water V_{Pk} required to completely fill up to category k, the volume of water $V_P - V_{Pk}$, and the depth to which this fills category $k+1$. Ice and snow areas a_i and a_s are also depicted. The volume calculation takes account of the presence of snow, which may be partially or completely saturated. (b) Schematic illustration indicating pond surface height $h_{pnd,tot}$ and sea level h_{sl} measured with respect to the thinnest surface height category h_{i1}, the submerged portion of the floe h_{sub}, and hydraulic head ΔH . A positive hydraulic head (pond surface above sea level) will flush melt water through the sea ice into the ocean; a negative hydraulic head can drive percolation of sea water onto the ice surface. Here, $\alpha = 0.6$ and $\beta = 0.4$ are the surface height and basal depth distribution fractions. The height of the steps is the height of the ice above the reference level, and the width of the steps is the area of ice of that height. The illustration does not imply a particular assumed topography, rather it is assumed that all thickness categories are present at the sub-grid scale so that water will always flow to the lowest surface height class.

Once a volume of water is produced from ice and snow melting, we calculate the number of ice categories covered by water. At each time step, we construct a list of volumes of water $\{V_{P1}, V_{P2}, ...V_{P,k-1}, V_{Pk}, V_{P,k+1}, ...\}$, where V_{Pk} is the volume of water required to completely cover the ice and snow in the surface height categories from $i = 1$ to $i = k$. The volume V_{Pk} is defined so that if the volume of water V_P is such that $V_{Pk} < V_P < V_{P,k+1}$ then the snow and ice in categories $n = 1$ to $n = k + 1$ are covered in melt water. Figure *Melt Ponds* (a) depicts the areas covered in melt water and saturated snow on the surface height (rather than thickness) categories $h_{top,k}$. Note in the code, we assume that $h_{top,n}/h_{in} = 0.6$ (an arbitrary choice). The fractional area of the nth category covered in snow is a_{sn}. The volume V_{P1}, which is the region with vertical hatching, is the volume of water required to completely fill the first thickness category, so that any extra melt water must occupy the second thickness category, and it is given by the expression

$$V_{P1} = a_{i1}(h_{top,2} - h_{top,1}) - a_{s1}a_{i1}h_{s1}(1 - V_{sw}), \tag{2.56}$$

where V_{sw} is the fraction of the snow volume that can be occupied by water, and h_{s1} is the snow depth on ice height class 1. In a similar way, the volume required to fill the first and second surface categories, V_{P2}, is given by

$$V_{P2} = a_{i1}(h_{top,3} - h_{top,2}) + a_{i2}(h_{top,3} - h_{top,2}) - a_{s2}a_{i2}h_{s2}(1 - V_{sw}) + V_{P1}. \tag{2.57}$$

The general expression for volume V_{Pk} is given by

$$V_{Pk} = \sum_{m=0}^{k} a_{im}(h_{top,k+1} - h_{top,k}) - a_{sk}a_{ik}h_{sk}(1 - V_{sw}) + \sum_{m=0}^{k-1} V_{Pm}. \tag{2.58}$$

(Note that we have implicitly assumed that $h_{si} < h_{top,k+1} - h_{top,k}$ for all k.) No melt water can be stored on the thickest ice thickness category. If the melt water volume exceeds the volume calculated above, the remaining melt water is released to the ocean.

At each time step, the pond height above the level of the thinnest surface height class, that is, the maximum pond depth, is diagnosed from the list of volumes V_{Pk}. In particular, if the total volume of melt water V_P is such that $V_{Pk} < V_P < V_{P,k+1}$ then the pond height $h_{pnd,tot}$ is

$$h_{pnd,tot} = h_{par} + h_{top,k} - h_{top,1}, \tag{2.59}$$

where h_{par} is the height of the pond above the level of the ice in class k and partially fills the volume between $V_{P,k}$ and $V_{P,k+1}$. From Figure *Melt Ponds* (a) we see that $h_{top,k} - h_{top,1}$ is the height of the melt water, which has volume V_{Pk}, which completely fills the surface categories up to category k. The remaining volume, $V_P - V_{Pk}$, partially fills category $k + 1$ to the height h_{par} and there are two cases to consider: either the snow cover on category $k + 1$, with height $h_{s,k+1}$, is completely covered in melt water (i.e., $h_{par} > h_{s,k+1}$), or it is not (i.e., $h_{par} \leq h_{s,k+1}$). From conservation of volume, we see from Figure *Melt Ponds* (a) that for an incompletely to completely saturated snow cover on surface ice class $k + 1$,

$$V_P - V_{Pk} = \quad h_{par}\left(\sum_{m=1}^{k} a_{ik} + a_{i,k+1}(1 - a_{s,k+1}) + a_{i,k+1}a_{s,k+1}V_{sw}\right) \quad \text{for} \quad h_{par} \leq h_{s,k+1}, \tag{2.60}$$

and for a saturated snow cover with water on top of the snow on surface ice class $k + 1$,

$$\begin{aligned} V_P - V_{Pk} = &\quad h_{par}\left(\sum_{m=1}^{k} a_{ik} + a_{i,k+1}(1 - a_{s,k+1})\right) + a_{i,k+1}a_{s,k+1}V_{sw}h_{s,k+1} \\ + &\quad a_{i,k+1}a_{s,k+1}(h_{par} - h_{s,k+1}) \quad \text{for} \quad h_{par} > h_{s,k+1}. \end{aligned} \tag{2.61}$$

As the melting season progresses, not only does melt water accumulate upon the upper surface of the sea ice, but the sea ice beneath the melt water becomes more porous owing to a reduction in solid fraction [13]. The hydraulic head of melt water on sea ice (i.e., its height above sea level) drives flushing of melt water through the porous sea ice and into the underlying ocean. The mushy thermodynamics scheme (*ktherm* = 2) handles flushing. For *ktherm* $\neq 2$ we model the vertical flushing rate using Darcy's law for flow through a porous medium

$$w = -\frac{\Pi_v}{\mu}\rho_o g\frac{\Delta H}{h_i}, \tag{2.62}$$

where w is the vertical mass flux per unit perpendicular cross-sectional area (i.e., the vertical component of the Darcy velocity), Π_v is the vertical component of the permeability tensor (assumed to be isotropic in the horizontal), μ is the viscosity of water, ρ_o is the ocean density, g is gravitational acceleration, ΔH is the the hydraulic head, and h_i is the thickness of the ice through which the pond flushes. As proposed by [20] the vertical permeability of sea ice can be calculated from the liquid fraction ϕ:

$$\Pi_v = 3 \times 10^{-8} \phi^3 \mathrm{m}^2. \tag{2.63}$$

Since the solid fraction varies throughout the depth of the sea ice, so does the permeability. The rate of vertical drainage is determined by the lowest (least permeable) layer, corresponding to the highest solid fraction. From the equations describing sea ice as a mushy layer [14], the solid fraction is determined by:

$$\phi = \frac{c_i - S}{c_i - S_{br}(T)}, \tag{2.64}$$

where S is the bulk salinity of the ice, $S_{br}(T)$ is the concentration of salt in the brine at temperature T and c_i is the concentration of salt in the ice crystals (set to zero).

The hydraulic head is given by the difference in height between the upper surface of the melt pond $h_{pnd,tot}$ and the sea level h_{sl}. The value of the sea level h_{sl} is calculated from

$$h_{sl} = h_{sub} - 0.4 \sum_{n=1}^{N} a_{in} h_{in} - \beta h_{i1}, \tag{2.65}$$

where $0.4 \sum_{n=1}^{N} a_{in} h_{i,n}$ is the mean thickness of the basal depth classes, and h_{sub} is the depth of the submerged portion of the floe. Figure *Melt Ponds* (b) depicts the relationship between the hydraulic head and the depths and heights that appear in Equation (2.65). The depth of the submerged portion of the floe is determined from hydrostatic equilibrium to be

$$h_{sub} = \frac{\rho_m}{\rho_w} V_P + \frac{\rho_s}{\rho_w} V_s + \frac{\rho_i}{\rho_w} V_i, \tag{2.66}$$

where ρ_m is the density of melt water, V_P is the total pond volume, V_s is the total snow volume, and V_i is the total ice volume.

When the surface energy balance is negative, a layer of ice is formed at the upper surface of the ponds. The rate of growth of the ice lid is given by the Stefan energy budget at the lid-pond interface

$$\rho_i L_0 \frac{dh_{ipnd}}{dt} = k_i \frac{\partial T_i}{\partial z} - k_p \frac{\partial T_p}{\partial z}, \tag{2.67}$$

where L_0 is the latent heat of fusion of pure ice per unit volume, T_i and T_p are the ice surface and pond temperatures, and k_i and k_p are the thermal conductivity of the ice lid and pond respectively. The second term on the right hand-side is close to zero since the pond is almost uniformly at the freezing temperature [67]. Approximating the temperature gradient in the ice lid as linear, the Stefan condition yields the classic Stefan solution for ice lid depth

$$h_{ipnd} = \sqrt{\frac{2k_i}{\rho_s L} \Delta T_i t}, \tag{2.68}$$

where ΔT is the temperature difference between the top and the bottom of the lid. Depending on the surface flux conditions the ice lid can grow, partially melt, or melt completely. Provided that the ice lid is thinner than a critical lid depth (1 cm is suggested) then the pond is regarded as effective, i.e. the pond affects the optical properties of the ice cover. Effective pond area and pond depth for each thickness category are passed to the radiation scheme for calculating albedo. Note that once the ice lid has exceeded the critical thickness, snow may accumulate on the lid causing a substantial increase in albedo. In the current CICE model, melt ponds only affect the thermodynamics of the ice through the albedo. To conserve energy, the ice lid is dismissed once the pond is completely refrozen.

As the sea ice area shrinks due to melting and ridging, the pond volume over the lost area is released to the ocean immediately. In [17], the pond volume was carried as an ice area tracer, but in [18] and here, pond area and thickness are carried as separate tracers, as in the *Tracers* section.

Unlike the cesm and level-ice melt pond schemes, the liquid pond water in the topo parameterization is not necessarily virtual; it can be withheld from being passed to the ocean model until the ponds drain by setting the namelist variable `l_mpond_fresh` = .true. The refrozen pond lids are still virtual. Extra code needed to track and enforce conservation of water has been added to **icepack_itd.F90** (subroutine *zap_small_areas*), **icepack_mechred.F90** (subroutine *ridge_shift*), **icepack_therm_itd.F90** (subroutines *linear_itd* and *lateral_melt*), and **icepack_therm_vertical.F90** (subroutine *thermo_vertical*), along with global diagnostics in **icedrv_diagnostics.F90**.

Level-ice formulation (`tr_pond_lvl` = true)

This meltpond parameterization represents a combination of ideas from the empirical CESM melt pond scheme and the topo approach, and is documented in [27]. The ponds evolve according to physically based process descriptions, assuming a thickness-area ratio for changes in pond volume. A novel aspect of the new scheme is that the ponds are carried as tracers on the level (undeformed) ice area of each thickness category, thus limiting their spatial extent based on the simulated sea ice topography. This limiting is meant to approximate the horizontal drainage of melt water into depressions in ice floes. (The primary difference between the level-ice and topo meltpond parameterizations lies in how sea ice topography is taken into account when determining the areal coverage of ponds.) Infiltration of the snow by melt water postpones the appearance of ponds and the subsequent acceleration of melting through albedo feedback, while snow on top of refrozen pond ice also reduces the ponds' effect on the radiation budget.

Melt pond processes, described in more detail below, include addition of liquid water from rain, melting snow and melting surface ice, drainage of pond water when its weight pushes the ice surface below sea level or when the ice interior becomes permeable, and refreezing of the pond water. If snow falls after a layer of ice has formed on the ponds, the snow may block sunlight from reaching the ponds below. When melt water forms with snow still on the ice, the water is assumed to infiltrate the snow. If there is enough water to fill the air spaces within the snowpack, then the pond becomes visible above the snow, thus decreasing the albedo and ultimately causing the snow to melt faster. The albedo also decreases as snow depth decreases, and thus a thin layer of snow remaining above a pond-saturated layer of snow will have a lower albedo than if the melt water were not present.

The level-ice formulation assumes a thickness-area ratio for *changes* in pond volume, while the CESM scheme assumes this ratio for the total pond volume. Pond volume changes are distributed as changes to the area and to the depth of the ponds using an assumed aspect ratio, or shape, given by the parameter δ_p (`pndaspect`), $\delta_p = \Delta h_p/\Delta a_p$ and $\Delta V = \Delta h_p \Delta a_p = \delta_p \Delta a_p^2 = \Delta h_p^2/\delta_p$. Here, $a_p = a_{pnd}a_{lvl}$, the mean pond area over the ice.

Given the ice velocity $\mathbf{u}$, conservation equations for level ice fraction $a_{lvl}a_i$, pond area fraction $a_{pnd}a_{lvl}a_i$, pond volume $h_{pnd}a_{pnd}a_{lvl}a_i$ and pond ice volume $h_{ipnd}a_{pnd}a_{lvl}a_i$ are

$$\frac{\partial}{\partial t}(a_{lvl}a_i) + \nabla\cdot(a_{lvl}a_i\mathbf{u}) = 0, \tag{2.69}$$

$$\frac{\partial}{\partial t}(a_{pnd}a_{lvl}a_i) + \nabla\cdot(a_{pnd}a_{lvl}a_i\mathbf{u}) = 0, \tag{2.70}$$

$$\frac{\partial}{\partial t}(h_{pnd}a_{pnd}a_{lvl}a_i) + \nabla\cdot(h_{pnd}a_{pnd}a_{lvl}a_i\mathbf{u}) = 0, \tag{2.71}$$

$$\frac{\partial}{\partial t}(h_{ipnd}a_{pnd}a_{lvl}a_i) + \nabla\cdot(h_{ipnd}a_{pnd}a_{lvl}a_i\mathbf{u}) = 0. \tag{2.72}$$

(We have dropped the category subscript here, for clarity.) Equations (2.71) and (2.72) express conservation of melt pond volume and pond ice volume, but in this form highlight that the quantities tracked in the code are the tracers h_{pnd} and h_{ipnd}, pond depth and pond ice thickness. Likewise, the level ice fraction a_{lvl} is a tracer on ice area fraction (Equation (2.69)), and pond fraction a_{pnd} is a tracer on level ice (Equation (2.70)).

Pond ice. The ponds are assumed to be well mixed fresh water, and therefore their temperature is 0°C. If the air temperature is cold enough, a layer of clear ice may form on top of the ponds. There are currently three options in the code for refreezing the pond ice. Only option A tracks the thickness of the lid ice using the tracer h_{ipnd} and includes the radiative effect of snow on top of the lid.

A. The `frzpnd` = 'hlid' option uses a Stefan approximation for growth of fresh ice and is invoked only when $\Delta V_{melt} = 0$.

The basic thermodynamic equation governing ice growth is

$$\rho_i L \frac{\partial h_i}{\partial t} = k_i \frac{\partial T_i}{\partial z} \sim k_i \frac{\Delta T}{h_i} \tag{2.73}$$

assuming a linear temperature profile through the ice thickness h_i. In discrete form, the solution is

$$\Delta h_i = \begin{cases} \sqrt{\beta \Delta t}/2 & \text{if } h_i = 0 \\ \beta \Delta t / 2h_i & \text{if } h_i > 0, \end{cases} \tag{2.74}$$

where

$$\beta = \frac{2k_i \Delta T}{\rho_i L}. \tag{2.75}$$

When $\Delta V_{melt} > 0$, any existing pond ice may also melt. In this case,

$$\Delta h_i = -\min\left(\frac{\max(F_\circ, 0)\Delta t}{\rho_i L}, h_i\right), \tag{2.76}$$

where $F_\circ$ is the net downward surface flux.

In either case, the change in pond volume associated with growth or melt of pond ice is

$$\Delta V_{frz} = -\Delta h_i a_{pnd} a_{lvl} a_i \rho_i / \rho_0, \tag{2.77}$$

where ρ_0 is the density of fresh water.

B. The `frzpnd` = 'cesm' option uses the same empirical function as in the CESM melt pond parameterization.

Radiative effects. Freshwater ice that has formed on top of a melt pond is assumed to be perfectly clear. Snow may accumulate on top of the pond ice, however, shading the pond and ice below. The depth of the snow on the pond ice is initialized as $h_{ps}^0 = F_{snow}\Delta t$ at the first snowfall after the pond ice forms. From that time until either the pond ice or the pond snow disappears, the pond snow depth tracks the depth of snow on sea ice (h_s) using a constant difference Δ. As h_s melts, $h_{ps} = h_s - \Delta$ will be reduced to zero eventually, at which time the pond ice is fully uncovered and shortwave radiation passes through.

To prevent a sudden change in the shortwave reaching the sea ice (which can prevent the thermodynamics from converging), thin layers of snow on pond ice are assumed to be patchy, thus allowing the shortwave flux to increase gradually as the layer thins. This is done using the same parameterization for patchy snow as is used elsewhere in Icepack, but with its own parameter h_{s1}:

$$a_{pnd}^{eff} = \left(1 - \min\left(h_{ps}/h_{s1}, 1\right)\right) a_{pnd} a_{lvl}. \tag{2.78}$$

If any of the pond ice melts, the radiative flux allowed to pass through the ice is reduced by the (roughly) equivalent flux required to melt that ice. This is accomplished (approximately) with $a_{pnd}^{eff} = (1 - f_{frac}) a_{pnd} a_{lvl}$, where (see Equation (2.76))

$$f_{frac} = \min\left(-\frac{\rho_i L \Delta h_i}{F_\circ \Delta t}, 1\right). \tag{2.79}$$

Snow infiltration by pond water. If there is snow on top of the sea ice, melt water may infiltrate the snow. It is a "virtual process" that affects the model's thermodynamics through the input parameters of the radiation scheme; it does not melt the snow or affect the snow heat content.

A snow pack is considered saturated when its percentage of liquid water content is greater or equal to 15% (Sturm and others, 2009). We assume that if the volume fraction of retained melt water to total liquid content

$$r_p = \frac{V_p}{V_p + V_s \rho_s / \rho_0} < 0.15, \tag{2.80}$$

then effectively there are no meltponds present, that is, $a_{pnd}^{eff} = h_{pnd}^{eff} = 0$. Otherwise, we assume that the snowpack is saturated with liquid water.

We assume that all of the liquid water accumulates at the base of the snow pack and would eventually melt the surrounding snow. Two configurations are therefore possible, (1) the top of the liquid lies below the snow surface and (2) the liquid water volume overtops the snow, and all of the snow is assumed to have melted into the pond. The volume of void space within the snow that can be filled with liquid melt water is

$$V_{mx} = h_{mx} a_p = \left(\frac{\rho_0 - \rho_s}{\rho_0} \right) h_s a_p, \tag{2.81}$$

and we compare V_p with V_{mx}.

Case 1: For $V_p < V_{mx}$, we define V_p^{eff} to be the volume of void space filled by the volume V_p of melt water: $\rho_0 V_p = (\rho_0 - \rho_s) V_p^{eff}$, or in terms of depths,

$$h_p^{eff} = \left(\frac{\rho_0}{\rho_0 - \rho_s} \right) h_{pnd}. \tag{2.82}$$

The liquid water under the snow layer is not visible and therefore the ponds themselves have no direct impact on the radiation ($a_{pnd}^{eff} = h_{pnd}^{eff} = 0$), but the effective snow thickness used for the radiation scheme is reduced to

$$h_s^{eff} = h_s - h_p^{eff} a_p = h_s - \frac{\rho_0}{\rho_0 - \rho_s} h_{pnd} a_p. \tag{2.83}$$

Here, the factor $a_p = a_{pnd} a_{lvl}$ averages the reduced snow depth over the ponds with the full snow depth over the remainder of the ice; that is, $h_s^{eff} = h_s(1 - a_p) + (h_s - h_p^{eff}) a_p$.

Case 2: Similarly, for $V_p \geq V_{mx}$, the total mass in the liquid is $\rho_0 V_p + \rho_s V_s = \rho_0 V_p^{eff}$, or

$$h_p^{eff} = \frac{\rho_0 h_{pnd} + \rho_s h_s}{\rho_0}. \tag{2.84}$$

Thus the effective depth of the pond is the depth of the whole slush layer h_p^{eff}. In this case, $a_{pnd}^{eff} = a_{pnd} a_{lvl}$.

Drainage. A portion $1 - r$ of the available melt water drains immediately into the ocean. Once the volume changes described above have been applied and the resulting pond area and depth calculated, the pond depth may be further reduced if the top surface of the ice would be below sea level or if the sea ice becomes permeable.

We require that the sea ice surface remain at or above sea level. If the weight of the pond water would push the mean ice–snow interface of a thickness category below sea level, some or all of the pond water is removed to bring the interface back to sea level via Archimedes' Principle written in terms of the draft d,

$$\rho_i h_i + \rho_s h_s + \rho_0 h_p = \rho_w d \leq \rho_w h_i. \tag{2.85}$$

There is a separate freeboard calculation in the thermodynamics which considers only the ice and snow and converts flooded snow to sea ice. Because the current melt ponds are "virtual" in the sense that they only have a radiative influence, we do not allow the pond mass to change the sea ice and snow masses at this time, although this issue may need to be reconsidered in the future, especially for the Antarctic.

The mushy thermodynamics scheme (*ktherm* = 2) handles flushing. For *ktherm* $\neq 2$, the permeability of the sea ice is calculated using the internal ice temperatures T_i (computed from the enthalpies as in the sea ice thermodynamics). The brine salinity and liquid fraction are given by [46] [eq 3.6] $S_{br} = 1/(10^{-3} - 0.054/T_i)$ and $\phi = S/S_{br}$, where S is the bulk salinity of the combined ice and brine. The ice is considered permeable if $\phi \geq 0.05$ with a permeability of $p = 3 \times 10^{-8} \min(\phi^3)$ (the minimum being taken over all of the ice layers). A hydraulic pressure head is computed as $P = g\rho_w \Delta h$ where Δh is the height of the pond and sea ice above sea level. Then the volume of water drained is given by

$$\Delta V_{perm} = -a_{pnd} \min\left(h_{pnd}, \frac{pPd_p \Delta t}{\mu h_i} \right), \tag{2.86}$$

where d_p is a scaling factor (dpscale), and $\mu = 1.79 \times 10^{-3}$ kg m $^{-1}$ s $^{-1}$ is the dynamic viscosity.

Conservation elsewhere. When ice ridges and when new ice forms in open water, the level ice area changes and ponds must be handled appropriately. For example, when sea ice deforms, some of the level ice is transformed into ridged ice. We assume that pond water (and ice) on the portion of level ice that ridges is lost to the ocean. All of the tracer volumes are altered at this point in the code, even though h_{pnd} and h_{ipnd} should not change; compensating factors in the tracer volumes cancel out (subroutine *ridge_shift* in **icepack_mechred.F90**).

When new ice forms in open water, level ice is added to the existing sea ice, but the new level ice does not yet have ponds on top of it. Therefore the fractional coverage of ponds on level ice decreases (thicknesses are unchanged). This is accomplished in **icepack_therm_itd.F90** (subroutine *add_new_ice*) by maintaining the same mean pond area in a grid cell after the addition of new ice,

$$a'_{pnd}(a_{lvl} + \Delta a_{lvl})(a_i + \Delta a_i) = a_{pnd} a_{lvl} a_i, \tag{2.87}$$

and solving for the new pond area tracer a'_{pnd} given the newly formed ice area $\Delta a_i = \Delta a_{lvl}$.

2.7.3 Thermodynamic surface forcing balance

The net surface energy flux from the atmosphere to the ice (with all fluxes defined as positive downward) is

$$F_0 = F_s + F_l + F_{L\downarrow} + F_{L\uparrow} + (1 - \alpha)(1 - i_0)F_{sw}, \tag{2.88}$$

where F_s is the sensible heat flux, F_l is the latent heat flux, $F_{L\downarrow}$ is the incoming longwave flux, $F_{L\uparrow}$ is the outgoing longwave flux, F_{sw} is the incoming shortwave flux, α is the shortwave albedo, and i_0 is the fraction of absorbed shortwave flux that penetrates into the ice. The albedo may be altered by the presence of melt ponds. Each of the explicit melt pond parameterizations (CESM, topo and level-ice ponds) should be used in conjunction with the Delta-Eddington shortwave scheme, described below.

Shortwave radiation: Delta-Eddington

Two methods for computing albedo and shortwave fluxes are available, the "ccsm3" method, described in the next section, and a multiple scattering radiative transfer scheme that uses a Delta-Eddington approach (`shortwave` = `dEdd`).

"Inherent" optical properties (IOPs) for snow and sea ice, such as extinction coefficient and single scattering albedo, are prescribed based on physical measurements; reflected, absorbed and transmitted shortwave radiation ("apparent" optical properties) are then computed for each snow and ice layer in a self-consistent manner. Absorptive effects of inclusions in the ice/snow matrix such as dust and algae can also be included, along with radiative treatment of melt ponds and other changes in physical properties, for example granularization associated with snow aging.

The Delta-Eddington formulation is described in detail in [8]. Since publication of this technical paper, a number of improvements have been made to the Delta-Eddington scheme, including a surface scattering layer and internal shortwave absorption for snow, generalization for multiple snow layers and more than four layers of ice, and updated IOP values.

In addition, a 5-band option for snow has been added based on [10] using parameters derived from the SNICAR snow model (`shortwave` = `dEdd_snicar_ad`). The 3-band Delta-Eddington data is still used for non-snow-covered surfaces. The 5-band option calculates snow radiative transfer properties for 1 visible and 4 near-infrared bands, and the reflection, absorption and transmission of direct and diffuse shorwave incidents are computed separately, thus removing the snow grain adjustment used in the 3-band Delta-Eddington scheme. Also, albedo and absorption of snow-covered sea ice are adjusted for solar zenith angles greater than 75 degrees. Because the 5-band lookup tables are very large, they can be slow to compile. The setting `ICE_SNICARHC` is false for simulations not using the `dEdd_snicar_ad` option, and must be set to true in order to use the hard-coded (HC) lookup tables generated from the SNICAR model.

The namelist parameters `R_ice` and `R_pnd` adjust the albedo of bare or ponded ice by the product of the namelist value and one standard deviation. For example, if `R_ice` = 0.1, the albedo increases by 0.1σ. Similarly, setting `R_snw` = 0.1

decreases the snow grain radius by 0.1σ (thus increasing the albedo). Two additional tuning parameters are available for this scheme, `dT_mlt` and `rsnw_mlt`. `dT_mlt` is the temperature change needed for a change in snow grain radius from non-melting to melting, and `rsnw_mlt` is the maximum snow grain radius when melting. An absorption coefficient for algae (`kalg`) may also be set. See [8] for details; the CESM melt pond and Delta-Eddington parameterizations are further explained and validated in [23].

Shortwave radiation: CCSM3

In the parameterization used in the previous version of the Community Climate System Model (CCSM3), the albedo depends on the temperature and thickness of ice and snow and on the spectral distribution of the incoming solar radiation. Albedo parameters have been chosen to fit observations from the SHEBA field experiment. For $T_{sf} < -1^\circ C$ and $h_i >$ *ahmax*, the bare ice albedo is 0.78 for visible wavelengths ($<$ 700 nm) and 0.36 for near IR wavelengths ($>$ 700 nm). As h_i decreases from ahmax to zero, the ice albedo decreases smoothly (using an arctangent function) to the ocean albedo, 0.06. The ice albedo in both spectral bands decreases by 0.075 as T_{sf} rises from $-1^\circ C$ to . The albedo of cold snow ($T_{sf} < -1^\circ C$) is 0.98 for visible wavelengths and 0.70 for near IR wavelengths. The visible snow albedo decreases by 0.10 and the near IR albedo by 0.15 as T_{sf} increases from $-1^\circ C$ to $0^\circ C$. The total albedo is an area-weighted average of the ice and snow albedos, where the fractional snow-covered area is

$$f_{snow} = \frac{h_s}{h_s + h_{snowpatch}}, \tag{2.89}$$

and $h_{snowpatch} = 0.02$ m. The envelope of albedo values is shown in Figure *Albedo*. This albedo formulation incorporates the effects of melt ponds implicitly; the explicit melt pond parameterization is not used in this case.

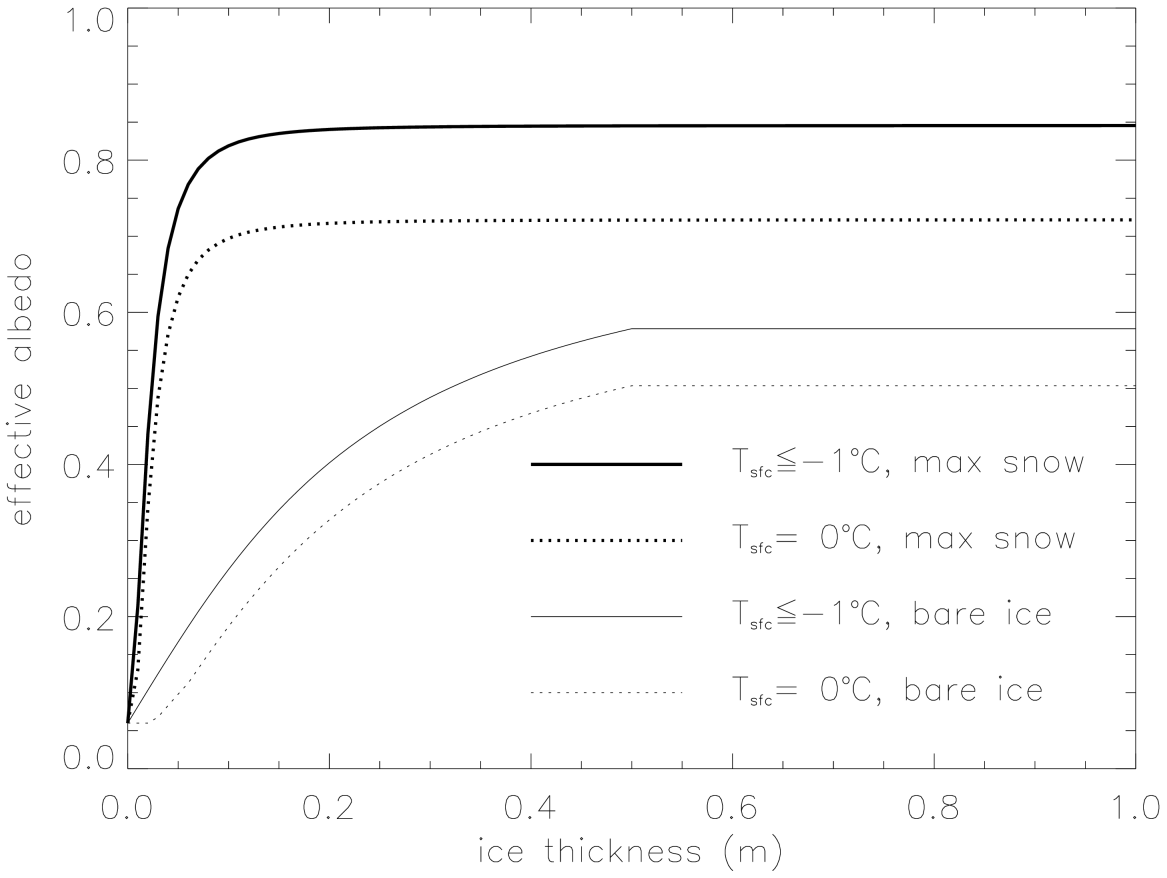

Fig. 4: *Albedo*

Figure *Albedo* illustrates Albedo as a function of ice thickness and temperature, for the two extrema in snow depth, for the ccsm3 shortwave option. Maximum snow depth is computed based on Archimedes' Principle for the given ice thickness. These curves represent the envelope of possible albedo values.

The net absorbed shortwave flux is $F_{swabs} = \sum(1-\alpha_j)F_{sw\downarrow}$, where the summation is over four radiative categories (direct and diffuse visible, direct and diffuse near infrared). The flux penetrating into the ice is $I_0 = i_0 \, F_{swabs}$, where $i_0 = 0.70 \, (1-f_{snow})$ for visible radiation and $i_0 = 0$ for near IR. Radiation penetrating into the ice is attenuated according to Beer's Law:

$$I(z) = I_0 \exp(-\kappa_i z), \tag{2.90}$$

where $I(z)$ is the shortwave flux that reaches depth z beneath the surface without being absorbed, and κ_i is the bulk extinction coefficient for solar radiation in ice, set to 1.4 m^{-1} for visible wavelengths [12]. A fraction $\exp(-\kappa_i h_i)$ of the penetrating solar radiation passes through the ice to the ocean ($F_{sw\Downarrow}$).

Longwave radiation, turbulent fluxes

While incoming shortwave and longwave radiation are obtained from the atmosphere, outgoing longwave radiation and the turbulent heat fluxes are derived quantities. Outgoing longwave takes the standard blackbody form, $F_{L\uparrow} = \epsilon\sigma \left(T_{sf}^K\right)^4$, where $\epsilon = 0.985$ is the emissivity of snow or ice, σ is the Stefan-Boltzmann constant and T_{sf}^K is the surface temperature in Kelvin. (The longwave fluxes are partitioned such that $\epsilon F_{L\downarrow}$ is absorbed at the surface, the remaining $(1-\epsilon)\, F_{L\downarrow}$ being returned to the atmosphere via $F_{L\uparrow}$.) The sensible heat flux is proportional to the difference between air potential temperature and the surface temperature of the snow or snow-free ice,

$$F_s = C_s \left(\Theta_a - T_{sf}^K\right). \tag{2.91}$$

C_s and C_l (below) are nonlinear turbulent heat transfer coefficients described in the *Atmosphere* section. Similarly, the latent heat flux is proportional to the difference between Q_a and the surface saturation specific humidity Q_{sf}:

$$\begin{aligned} F_l = & \quad C_l \left(Q_a - Q_{sf}\right), \\ Q_{sf} = & \quad (q_1/\rho_a) \exp(-q_2/T_{sf}^K), \end{aligned}$$

where $q_1 = 1.16378 \times 10^7 \,\text{kg/m}^3$, $q_2 = 5897.8 \,\text{K}$, T_{sf}^K is the surface temperature in Kelvin, and ρ_a is the surface air density.

The net downward heat flux from the ice to the ocean is given by [43]:

$$F_{bot} = -\rho_w c_w c_h u_* (T_w - T_f), \tag{2.92}$$

where ρ_w is the density of seawater, c_w is the specific heat of seawater, $c_h = 0.006$ is a heat transfer coefficient, $u_* = \sqrt{\left|\vec{\tau}_w\right| / \rho_w}$ is the friction velocity, and T_w is the sea surface temperature. A minimum value of u_* is available; we recommend $u_{*\,\min} = 5 \times 10^{-4}$ m/s, but the optimal value may depend on the ocean forcing used and can be as low as 0.

F_{bot} is limited by the total amount of heat available from the ocean, F_{frzmlt}. Additional heat, F_{side}, is used to melt the ice laterally following [44] and [64]. F_{bot} and the fraction of ice melting laterally are scaled so that $F_{bot} + F_{side} \geq F_{frzmlt}$ in the case that $F_{frzmlt} < 0$ (melting; see *Growth and melting*).

2.7.4 New temperatures

Bitz and Lipscomb thermodynamics (`ktherm` = 1)

The "Bitz99" thermodynamic sea ice model is based on [45] and [6], and is described more fully in [37]. The vertical salinity profile is prescribed and is unchanging in time. The snow is assumed to be fresh, and the midpoint salinity S_{ik} in each ice layer is given by

$$S_{ik} = \frac{1}{2} S_{\max}[1 - \cos(\pi z^{(\frac{a}{z+b})})], \tag{2.93}$$

where $z \equiv (k - 1/2)/N_i$, $S_{\max} = 3.2$ ppt, and $a = 0.407$ and $b = 0.573$ are determined from a least-squares fit to the salinity profile observed in multiyear sea ice by [60]. This profile varies from $S = 0$ at the top surface ($z = 0$) to $S = S_{\max}$ at the bottom surface ($z = 1$) and is similar to that used by [45]. Equation (2.93) is fairly accurate for ice that has drained at the top surface due to summer melting. It is not a good approximation for cold first-year ice, which has a more vertically uniform salinity because it has not yet drained. However, the effects of salinity on heat capacity are small for temperatures well below freezing, so the salinity error does not lead to significant temperature errors.

Temperature updates

Given the temperatures T_{sf}^m, T_s^m, and T_{ik}^m at time m, we solve a set of finite-difference equations to obtain the new temperatures at time $m + 1$. Each temperature is coupled to the temperatures of the layers immediately above and below by heat conduction terms that are treated implicitly. For example, the rate of change of T_{ik} depends on the new temperatures in layers $k - 1$, k, and $k + 1$. Thus we have a set of equations of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.94}$$

where $\mathbf{A}$ is a tridiagonal matrix, $\mathbf{x}$ is a column vector whose components are the unknown new temperatures, and $\mathbf{b}$ is another column vector. Given $\mathbf{A}$ and $\mathbf{b}$, we can compute $\mathbf{x}$ with a standard tridiagonal solver.

There are four general cases: (1) $T_{sf} < 0°C$, snow present; (2) $T_{sf} = 0°C$, snow present; (3) $T_{sf} < 0°C$, snow absent; and (4) $T_{sf} = 0°C$, snow absent. For case 1 we have one equation (the top row of the matrix) for the new surface temperature, N_s equations for the new snow temperatures, and N_i equations for the new ice temperatures. For cases 2 and 4 we omit the equation for the surface temperature, which is held at $0°C$, and for cases 3 and 4 we omit the snow temperature equations. Snow is considered absent if the snow depth is less than a user-specified minimum value, `hs_min`. (Very thin snow layers are still transported conservatively by the transport modules; they are simply ignored by the thermodynamics.)

The rate of temperature change in the ice interior is given by [45]:

$$\rho_i c_i \frac{\partial T_i}{\partial t} = \frac{\partial}{\partial z}\left(K_i \frac{\partial T_i}{\partial z}\right) - \frac{\partial}{\partial z}[I_{pen}(z)], \tag{2.95}$$

where $\rho_i = 917\ \mathrm{kg/m^3}$ is the sea ice density (assumed to be uniform), $c_i(T, S)$ is the specific heat of sea ice, $K_i(T, S)$ is the thermal conductivity of sea ice, I_{pen} is the flux of penetrating solar radiation at depth z, and z is the vertical coordinate, defined to be positive downward with $z = 0$ at the top surface. If `shortwave` = 'ccsm3', the penetrating radiation is given by Beer's Law:

$$I_{pen}(z) = I_0 \exp(-\kappa_i z),$$

where I_0 is the penetrating solar flux at the top ice surface and κ_i is an extinction coefficient. If `shortwave` = 'dEdd', then solar absorption is computed by the Delta-Eddington scheme.

The specific heat of sea ice is given to an excellent approximation by [48]

$$c_i(T, S) = c_0 + \frac{L_0 \mu S}{T^2}, \tag{2.96}$$

where $c_0 = 2106$ J/kg/deg is the specific heat of fresh ice at , $L_0 = 3.34 \times 10^5$ J/kg is the latent heat of fusion of fresh ice at , and $\mu = 0.054$ deg/ppt is the (liquidus) ratio between the freezing temperature and salinity of brine.

Following [72] and [45], the standard thermal conductivity (`conduct` = 'Maykut71') is given by

$$K_i(T,S) = K_0 + \frac{\beta S}{T}, \tag{2.97}$$

where $K_0 = 2.03$ W/m/deg is the conductivity of fresh ice and $\beta = 0.13$ W/m/ppt is an empirical constant. Experimental results [69] suggest that Equation (2.97) may not be a good description of the thermal conductivity of sea ice. In particular, the measured conductivity does not markedly decrease as T approaches $0°C$, but does decrease near the top surface (regardless of temperature).

An alternative parameterization based on the "bubbly brine" model of [50] for conductivity is available (`conduct` = 'bubbly'):

$$K_i = \frac{\rho_i}{\rho_0}\left(2.11 - 0.011T + 0.09S/T\right), \tag{2.98}$$

where ρ_i and $\rho_0 = 917$ kg/m^3 are densities of sea ice and pure ice. Whereas the parameterization in Equation (2.97) asymptotes to a constant conductivity of 2.03 W m^{-1} K $^{-1}$ with decreasing T, K_i in Equation (2.98) continues to increase with colder temperatures.

The equation for temperature changes in snow is analogous to Equation (2.95), with $\rho_s = 330$ kg/m^3, $c_s = c_0$, and $K_s = 0.30$ W/m/deg replacing the corresponding ice values. If shortwave = 'ccsm3', then the penetrating solar radiation is equal to zero for snow-covered ice, since most of the incoming sunlight is absorbed near the top surface. If shortwave = 'dEdd', however, then I_{pen} is nonzero in snow layers.

It is possible that more shortwave penetrates into an ice layer than is needed to completely melt the layer, or else it causes the computed temperature to be greater than the melting temperature, which until now has caused the vertical thermodynamics code to abort. A parameter `frac` = 0.9 sets the fraction of the ice layer than can be melted through. A minimum temperature difference for absorption of radiation is also set, currently `dTemp` = 0.02 (K). The limiting occurs in **icepack_therm_vertical.F90**, for both the ccsm3 and delta Eddington radiation schemes. If the available energy would melt through a layer, then penetrating shortwave is first reduced, possibly to zero, and if that is insufficient then the local conductivity is also reduced to bring the layer temperature just to the melting point.

We now convert Equation (2.95) to finite-difference form. The resulting equations are second-order accurate in space, except possibly at material boundaries, and first-order accurate in time. Before writing the equations in full we give finite-difference expressions for some of the terms.

First consider the terms on the left-hand side of Equation (2.95). We write the time derivatives as

$$\frac{\partial T}{\partial t} = \frac{T^{m+1} - T^m}{\Delta t},$$

where T is the temperature of either ice or snow and m is a time index. The specific heat of ice layer k is approximated as

$$c_{ik} = c_0 + \frac{L_0 \mu S_{ik}}{T_{ik}^m T_{ik}^{m+1}}, \tag{2.99}$$

which ensures that energy is conserved during a change in temperature. This can be shown by using Equation (2.96) to integrate $c_i\, dT$ from T_{ik}^m to T_{ik}^{m+1}; the result is $c_{ik}(T_{ik}^{m+1} - T_{ik}^m)$, where c_{ik} is given by Equation (2.99). The specific heat is a nonlinear function of T_{ik}^{m+1}, the unknown new temperature. We can retain a set of linear equations, however, by initially guessing $T_{ik}^{m+1} = T_{ik}^m$ and then iterating the solution, updating T_{ik}^{m+1} in Equation (2.99) with each iteration until the solution converges.

Next consider the first term on the right-hand side of Equation (2.95). The first term describes heat diffusion and is discretized for a given ice or snow layer k as

$$\frac{\partial}{\partial z}\left(K\frac{\partial T}{\partial z}\right) = \frac{1}{\Delta h}\left[K_k^*(T_{k-1}^{m+1} - T_k^{m+1}) - K_{k+1}^*(T_k^{m+1} - T_{k+1}^{m+1})\right], \tag{2.100}$$

where Δh is the layer thickness and K_k is the effective conductivity at the upper boundary of layer k. This discretization is centered and second-order accurate in space, except at the boundaries. The flux terms on the right-hand side (RHS) are treated implicitly; i.e., they depend on the temperatures at the new time $m+1$. The resulting scheme is first-order accurate in time and unconditionally stable. The effective conductivity K^* at the interface of layers $k-1$ and k is defined as

$$K_k^* = \frac{2K_{k-1}K_k}{K_{k-1}h_k + K_k h_{k-1}},$$

which reduces to the appropriate values in the limits $K_k \gg K_{k-1}$ (or vice versa) and $h_k \gg h_{k-1}$ (or vice versa). The effective conductivity at the top (bottom) interface of the ice-snow column is given by $K^* = 2K/\Delta h$, where K and Δh are the thermal conductivity and thickness of the top (bottom) layer. The second term on the RHS of Equation (2.95) is discretized as

$$\frac{\partial}{\partial z}\left[I_{pen}(z)\right] = I_0 \frac{\tau_{k-1} - \tau_k}{\Delta h} = \frac{I_k}{\Delta h}$$

where τ_k is the fraction of the penetrating solar radiation I_0 that is transmitted through layer k without being absorbed.

We now construct a system of equations for the new temperatures. For $T_{sf} < 0°C$ we require

$$F_0 = F_{ct}, \tag{2.101}$$

where F_{ct} is the conductive flux from the top surface to the ice interior, and both fluxes are evaluated at time $m+1$. Although F_0 is a nonlinear function of T_{sf}, we can make the linear approximation

$$F_0^{m+1} = F_0^* + \left(\frac{dF_0}{dT_{sf}}\right)^* (T_{sf}^{m+1} - T_{sf}^*),$$

where T_{sf}^* is the surface temperature from the most recent iteration, and F_0^* and $(dF_0/dT_{sf})^*$ are functions of T_{sf}^*. We initialize $T_{sf}^* = T_{sf}^m$ and update it with each iteration. Thus we can rewrite Equation (2.101) as

$$F_0^* + \left(\frac{dF_0}{dT_{sf}}\right)^* (T_{sf}^{m+1} - T_{sf}^*) = K_1^*(T_{sf}^{m+1} - T_1^{m+1}),$$

Rearranging terms, we obtain

$$\left[\left(\frac{dF_0}{dT_{sf}}\right)^* - K_1^*\right] T_{sf}^{m+1} + K_1^* T_1^{m+1} = \left(\frac{dF_0}{dT_{sf}}\right)^* T_{sf}^* - F_0^*, \tag{2.102}$$

the first equation in the set of equations (2.94). The temperature change in ice/snow layer k is

$$\rho_k c_k \frac{(T_k^{m+1} - T_k^m)}{\Delta t} = \frac{1}{\Delta h_k}[K_k^*(T_{k-1}^{m+1} - T_k^{m+1}) - K_{k+1}(T_k^{m+1} - T_{k+1}^{m+1})], \tag{2.103}$$

where $T_0 = T_{sf}$ in the equation for layer 1. In tridiagonal matrix form, Equation (2.103) becomes

$$-\eta_k K_k T_{k-1}^{m+1} + [1 + \eta_k(K_k + K_{k+1})]\, T_k^{m+1} - \eta_k K_{k+1} T_{k+1}^{m+1} = T_k^m + \eta_k I_k, \tag{2.104}$$

where $\eta_k = \Delta t/(\rho_k c_k \Delta h_k)$. In the equation for the bottom ice layer, the temperature at the ice–ocean interface is held fixed at T_f, the freezing temperature of the mixed layer; thus the last term on the LHS is known and is moved to the RHS. If $T_{sf} = 0°C$, then there is no surface flux equation. In this case the first equation in Equation (2.94) is similar to Equation (2.104), but with the first term on the LHS moved to the RHS.

These equations are modified if T_{sf} and F_{ct} are computed within the atmospheric model and passed to the host sea ice model (`calc_Tsfc` = false; see *Atmosphere*). In this case there is no surface flux equation. The top layer temperature is computed by an equation similar to Equation (2.104) but with the first term on the LHS replaced by $\eta_1 F_{ct}$ and moved to the RHS. The main drawback of treating the surface temperature and fluxes explicitly is that the solution scheme is no longer unconditionally stable. Instead, the effective conductivity in the top layer must satisfy a diffusive CFL condition:

$$K^* \le \frac{\rho c h}{\Delta t}.$$

For thin layers and typical coupling intervals ($\sim$ 1 hr), K^* may need to be limited before being passed to the atmosphere via the coupler. Otherwise, the fluxes that are returned to the host sea ice model may result in oscillating, highly inaccurate temperatures. The effect of limiting is to treat the ice as a poor heat conductor. As a result, winter growth rates are reduced, and the ice is likely to be too thin (other things being equal). The values of `hs_min` and Δt must therefore be chosen with care. If `hs_min` is too small, frequent limiting is required, but if `hs_min` is too large, snow will be ignored when its thermodynamic effects are significant. Likewise, infrequent coupling requires more limiting, whereas frequent coupling is computationally expensive.

This completes the specification of the matrix equations for the four cases. We compute the new temperatures using a tridiagonal solver. After each iteration we check to see whether the following conditions hold:

1. $T_{sf} \leq 0°C$.
2. The change in T_{sf} since the previous iteration is less than a prescribed limit, $\Delta T_{\max}$.
3. $F_0 \geq F_{ct}$. (If $F_0 < F_{ct}$, ice would be growing at the top surface, which is not allowed.)
4. The rate at which energy is added to the ice by the external fluxes equals the rate at which the internal ice energy is changing, to within a prescribed limit $\Delta F_{\max}$.

We also check the convergence rate of T_{sf}. If T_{sf} is oscillating and failing to converge, we average temperatures from successive iterations to improve convergence. When all these conditions are satisfied—usually within two to four iterations for $\Delta T_{\max} \approx 0.01°C$ and $\Delta F_{max} \approx 0.01$ W/m^2—the calculation is complete.

To compute growth and melt rates (*Growth and melting*), we derive expressions for the enthalpy q. The enthalpy of snow (or fresh ice) is given by

$$q_s(T) = -\rho_s(-c_0 T + L_0).$$

Sea ice enthalpy is more complex, because of brine pockets whose salinity varies inversely with temperature. Since the salinity is prescribed, there is a one-to-one relationship between temperature and enthalpy. The specific heat of sea ice, given by Equation (2.96), includes not only the energy needed to warm or cool ice, but also the energy used to freeze or melt ice adjacent to brine pockets. Equation (2.96) can be integrated to give the energy δ_e required to raise the temperature of a unit mass of sea ice of salinity S from T to T':

$$\delta_e(T, T') = c_0(T' - T) + L_0 \mu S \left(\frac{1}{T} - \frac{1}{T'}\right).$$

If we let $T' = T_m \equiv -\mu S$, the temperature at which the ice is completely melted, we have

$$\delta_e(T, T_m) = c_0(T_m - T) + L_0 \left(1 - \frac{T_m}{T}\right).$$

Multiplying by ρ_i to change the units from J/kg to J/m^3 and adding a term for the energy needed to raise the meltwater temperature to , we obtain the sea ice enthalpy:

$$q_i(T, S) = -\rho_i \left[c_0(T_m - T) + L_0 \left(1 - \frac{T_m}{T}\right) - c_w T_m.\right] \tag{2.105}$$

Note that Equation (2.105) is a quadratic equation in T. Given the layer enthalpies we can compute the temperatures using the quadratic formula:

$$T = \frac{-b - \sqrt{b^2 - 4ac}}{2a},$$

where

$$\begin{aligned} a = & \quad c_0, \\ b = & \quad (c_w - c_0)\, T_m - \frac{q_i}{\rho_i} - L_0, \\ c = & \quad L_0 T_m. \end{aligned}$$

The other root is unphysical.

Mushy thermodynamics (`ktherm` = 2)

The “mushy” thermodynamics option treats the sea ice as a mushy layer [14] in which the ice is assumed to be composed of microscopic brine inclusions surrounded by a matrix of pure water ice. Both enthalpy and salinity are prognostic variables. The size of the brine inclusions is assumed to be much smaller than the size of the ice layers, allowing a continuum approximation: a bulk sea-ice quantity is taken to be the liquid-fraction-weighted average of that quantity in the ice and in the brine.

Enthalpy and mushy physics

The mush enthalpy, q, is related to the temperature, T, and the brine volume, ϕ, by

$$q = \phi q_{br} \quad + (1-\phi) q_i = \phi \rho_w c_w T \quad + (1-\phi)(\rho_i c_i T - \rho_i L_0) \tag{2.106}$$

where q_{br} is the brine enthalpy, q_i is the pure ice enthalpy, ρ_i and c_i are density and heat capacity of the ice, ρ_w and c_w are density and heat capacity of the brine and L_0 is the latent heat of melting of pure ice. We assume that the specific heats of the ice and brine are fixed at the values of cp_ice and cp_ocn, respectively. The enthalpy is the energy required to raise the temperature of the sea ice to $0°C$, including both sensible and latent heat changes. Since the sea ice contains salt, it usually will be fully melted at a temperature below $0°C$. Equations (2.105) and (2.106) are equivalent except for the density used in the term representing the energy required to bring the melt water temperature to $0°C$ (ρ_i and ρ_w in equations (2.105) and (2.106), respectively).

The liquid fraction, ϕ, of sea ice is given by

$$\phi = \frac{S}{S_{br}}$$

where the brine salinity, S_{br}, is given by the liquidus relation using the ice temperature.

Within the parameterizations of brine drainage the brine density is a function of brine salinity [46]:

$$\rho(S_{br}) = 1000.3 + 0.78237 S_{br} + 2.8008 \times 10^{-4} S_{br}^2.$$

Outside the parameterizations of brine drainage the densities of brine and ice are fixed at the values of ρ_w and ρ_i, respectively.

The permeability of ice is computed from the liquid fraction as in [20]:

$$\Pi(\phi) = 3 \times 10^{-8} (\phi - \phi_\Pi)^3$$

where ϕ_Π is 0.05.

The liquidus relation used in the mushy layer module is based on observations of [4]. A piecewise linear relation can be fitted to observations of Z (the ratio of mass of salt (in g) to mass of pure water (in kg) in brine) to the melting temperature: $Z = aT + b$. Salinity is the mass of salt (in g) per mass of brine (in kg) so is related to Z by

$$\frac{1}{S} = \frac{1}{1000} + \frac{1}{Z}.$$

The data is well fitted with two linear regions,

$$S_{br} = \frac{(T + J_1)}{(T/1000 + L_1)} l_0 + \frac{(T + J_2)}{(T/1000 + L_2)} (1 - l_0)$$

where

$$l_0 = \begin{cases} 1 & \text{if} \quad T \geq T_0 \\ 0 & \text{if} \quad T < T_0 \end{cases},$$

$$J_{1,2} = \frac{b_{1,2}}{a_{1,2}},$$

$$L_{1,2} = \frac{(1 + b_{1,2}/1000)}{a_{1,2}}.$$

T_0 is the temperature at which the two linear regions meet. Fitting to the data, $T_0 = -7.636°\mathrm{C}$, $a_1 = -18.48$ g kg^{-1} K^{-1}, $a_2 = -10.3085$ g kg^{-1} K^{-1}, $b_1 = 0$ and $b_2 = 62.4$ g kg^{-1}.

Two-stage outer iteration

As for the Bitz99 thermodynamics [6] there are two qualitatively different situations that must be considered when solving for the vertical thermodynamics: the surface can be melting and at the melting temperature, or the surface can be colder than the melting temperature and not melting. In the Bitz99 thermodynamics these two situations were treated within the same iterative loop, but here they are dealt with separately. If at the beginning of the time step the ice surface is cold and not melting, we solve the ice temperatures assuming that this is also true at the end of the time step. Once we have solved for the new temperatures we test to see if the answer is consistent with this assumption. If the surface temperature is below the melting temperature then we have found the appropriate consistent solution. If the surface is above the melting temperature at the end of the initial solution attempt, we recalculate the new temperatures assuming the surface temperature is fixed at the melting temperature. Alternatively if the surface is at the melting temperature at the start of a time step, we assume initially that this is also the case at the end of the time step, solve for the new temperatures and then check that the surface conductive heat flux is less than the surface atmospheric heat flux as is required for a melting surface. If this is not the case, the temperatures are recalculated assuming the surface is colder than melting. We have found that solutions of the temperature equations that only treat one of the two qualitatively different solutions at a time are more numerically robust than if both are solved together. The surface state rarely changes qualitatively during the solution so the method is also numerically efficient.

Temperature updates

During the calculation of the new temperatures and salinities, the liquid fraction is held fixed at the value from the previous time step. Updating the liquid fraction during the Picard iteration described below was found to be numerically unstable. Keeping the liquid fraction fixed drastically improves the numerical stability of the method without significantly changing the solution.

Temperatures are calculated in a similar way to Bitz99 with an outer Picard iteration of an inner tridiagonal matrix solve. The conservation equation for the internal ice temperatures is

$$\frac{\partial q}{\partial t} = \frac{\partial}{\partial z}\left(K\frac{\partial T}{\partial z}\right) + w\frac{\partial q_{br}}{\partial z} + F$$

where q is the sea ice enthalpy, K is the bulk thermal conductivity of the ice, w is the vertical Darcy velocity of the brine, q_{br} is the brine enthalpy and F is the internally absorbed shortwave radiation. The first term on the right represents heat conduction and the second term represents the vertical advection of heat by gravity drainage and flushing.

The conductivity of the mush is given by

$$K = \phi K_{br} + (1-\phi)K_i$$

where $K_i = 2.3\mathrm{Wm}^{-1}\mathrm{K}^{-1}$ is the conductivity of pure ice and $K_{br} = 0.5375\mathrm{Wm}^{-1}\mathrm{K}^{-1}$ is the conductivity of the brine. The thermal conductivity of brine is a function of temperature and salinity, but here we take it as a constant value for the middle of the temperature range experienced by sea ice, $-10°$C [63], assuming the brine liquidus salinity at $-10°$C.

We discretize the terms that include temperature in the heat conservation equation as

$$\frac{q_k^t - q_k^{t_0}}{\Delta t} = \frac{\frac{K^*_{k+1}}{\Delta z'_{k+1}}(T^t_{k+1} - T^t_k) - \frac{K^*_k}{\Delta z'_k}(T^t_k - T^t_{k-1})}{\Delta h} \tag{2.107}$$

where the superscript signifies whether the quantity is evaluated at the start (t_0) or the end (t) of the time step and the subscript indicates the vertical layer. Writing out the temperature dependence of the enthalpy term we have

$$\frac{(\phi(c_w\rho_w - c_i\rho_i) + c_i\rho_i)\,T^t_k - (1-\phi)\rho_i L - q_k^{t_0}}{\Delta t} = \frac{\frac{K^*_{k+1}}{\Delta z'_{k+1}}(T^t_{k+1} - T^t_k) - \frac{K^*_k}{\Delta z'_k}(T^t_k - T^t_{k-1})}{\Delta h}.$$

The mush thermal conductivities are fixed at the start of the timestep. For the lowest ice layer T_{k+1} is replaced with T_{bot}, the temperature of the ice base. Δh is the layer thickness and z'_k is the distance between the $k-1$ and k layer centers.

Similarly, for the snow layer temperatures we have the following discretized equation:

$$\frac{c_i \rho_s T_k^t - \rho_s L_0 - q_k^{t_0}}{\Delta t} = \frac{\frac{K_{k+1}^*}{\Delta z'_{k+1}}(T_{k+1}^t - T_k^t) - \frac{K_k^*}{\Delta z'_k}(T_k^t - T_{k-1}^t)}{\Delta h}.$$

For the upper-most layer (either ice layer or snow layer if it present) T_{k-1} is replaced with T_{sf}, the temperature of the surface.

If the surface is colder than the melting temperature then we also have to solve for the surface temperature, T_{sf}. Here we follow the methodology of Bitz99 described above.

These discretized temperature equations form a tridiagional matrix for the new temperatures and are solved with a standard tridiagonal solver. A Picard iteration is used to incorporate nonlinearity in the equations. The surface heat flux is a function of surface temperature and with each iteration, the surface heat flux is calculated with the new surface temperature until convergence is achieved. Convergence normally occurs after a few iterations once the temperature changes during an iteration fall below 5×10^{-4} °C and the energy conservation error falls below 0.9 `ferrmax`.

Salinity updates

Several physical processes alter the sea ice bulk salinity. New ice forms with the salinity of the sea water from which it formed. Gravity drainage reduces the bulk salinity of newly formed sea ice, while flushing of melt water through the ice also alters the salinity profile.

The salinity equation takes the form

$$\frac{\partial S}{\partial t} = w \frac{\partial S_{br}}{\partial z} + G$$

where w is a vertical Darcy velocity and G is a source term. The right-hand side depends indirectly on the bulk salinity through the liquid fraction ($S = \phi S_{br}$). Since ϕ is fixed for the time step, we solve the salinity equation explicitly after the temperature equation is solved.

A. Gravity drainage. Sea ice initially retains all the salt present in the sea water from which it formed. Cold temperatures near the top surface of forming sea ice result in higher brine salinities there, because the brine is always at its melting temperature. This colder, saltier brine is denser than the underlying sea water and the brine undergoes convective overturning with the ocean. As the dense, cold brine drains out of the ice, it is replaced by fresher seawater, lowering the bulk salinity of the ice. Following [71], gravity drainage is assumed to occur as two simultaneously operating modes: a rapid mode operating principally near the ice base and a slow mode occurring everywhere.

Rapid drainage takes the form of a vertically varying upward Darcy flow. The contribution to the bulk salinity equation for the rapid mode is

$$\left.\frac{\partial S}{\partial t}\right|_{rapid} = w(z) \frac{\partial S_{br}}{\partial z}$$

where S is the bulk salinity and B_{br} is the brine salinity, specified by the liquidus relation with ice temperature. This equation is discretized using an upwind advection scheme,

$$\frac{S_k^t - S_k^{t_0}}{\Delta t} = w_k \frac{S_{brk+1} - S_{brk}}{\Delta z}.$$

The upward advective flow also carries heat, contributing a term to the heat conservation Equation (2.107),

$$\left.\frac{\partial q}{\partial t}\right|_{rapid} = w(z) \frac{\partial q_{br}}{\partial z}$$

where q_{br} is the brine enthalpy. This term is discretized as

$$\left.\frac{q_k^t - q_k^{t_0}}{\Delta t}\right|_{rapid} = w_k \frac{q_{br\,k+1} - q_{br\,k}}{\Delta z}.$$

$$w_k = \max_{j=k,n} (\tilde{w}_j)$$

where the maximum is taken over all the ice layers between layer k and the ice base. $\tilde{w}_j$ is given by

$$\tilde{w}(z) = w \left(\frac{Ra(z) - Ra_c}{Ra(z)} \right). \tag{2.108}$$

where Ra_c is a critical Rayleigh number and $Ra(z)$ is the local Rayleigh number at a particular level,

$$Ra(z) = \frac{g \Delta\rho \Pi (h - z)}{\kappa \eta}$$

where $\Delta\rho$ is the difference in density between the brine at z and the ocean, Π is the minimum permeability between z and the ocean, h is the ice thickness, κ is the brine thermal diffusivity and η is the brine dynamic viscosity. Equation (2.108) reduces the flow rate for Rayleigh numbers below the critical Rayleigh number.

The unmodified flow rate, w, is determined from a hydraulic pressure balance argument for upward flow through the mush and returning downward flow through ice free channels:

$$w(z)\Delta x^2 = A_m \left(-\frac{\Delta P}{l} + B_m \right)$$

where

$$\begin{aligned} \frac{\Delta P}{l} &= \frac{A_p B_p + A_m B_m}{A_m + A_p}, \\ A_m &= \frac{\Delta x^2}{\eta} \frac{n}{\sum_{k=1}^{n} \frac{1}{\Pi(k)}}, \\ B_m &= -\frac{g}{n} \sum_{k=1}^{n} \rho(k), \\ A_p &= \frac{\pi a^4}{8\eta}, \\ B_p &= -\rho_p g. \end{aligned}$$

There are three tunable parameters in the above parameterization, a, the diameter of the channel, Δx, the horizontal size of the mush draining through each channel, and Ra_c, the critical Rayleigh number. ρ_p is the density of brine in the channel which we take to be the density of brine in the mush at the level that the brine is draining from. l is the thickness of mush from the ice base to the top of the layer in question. We assume that Δx is proportional to l so that $\Delta x = 2\beta l$. a (`a_rapid_mode`), β (`aspect_rapid_mode`) and Ra_c (`Ra_c_rapid_mode`) are all namelist parameters with default values of 0.5 mm, 1 and 10, respectively. The value $\beta = 1$ gives a square aspect ratio for the convective flow in the mush.

The *slow drainage* mode takes the form of a simple relaxation of bulk salinity:

$$\left.\frac{\partial S(z)}{\partial t}\right|_{slow} = -\lambda (S(z) - S_c).$$

The decay constant, λ, is modeled as

$$\lambda = S^* \max \left(\frac{T_{bot} - T_{sf}}{h}, 0 \right)$$

where S^* is a tuning parameter for the drainage strength, T_{bot} is the basal ice temperature, T_{sf} is the upper surface temperature and h is the ice thickness. The bulk salinity relaxes to a value, $S_c(z)$, given by

$$S_c(z) = \phi_c S_{br}(z)$$

where $S_{br}(z)$ is the brine salinity at depth z and ϕ_c is a critical liquid fraction. Both S^* and ϕ_c are namelist parameters, `dSdt_slow_mode` $= 1.5 \times 10^{-7}$ m s^{-1} K^{-1} and `phi_c_slow_mode` $= 0.05$.

B. Downwards flushing. Melt pond water drains through sea ice and flushes out brine, reducing the bulk salinity of the sea ice. This is modeled with the mushy physics option as a vertical Darcy flow through the ice that affects both the enthalpy and bulk salinity of the sea ice:

$$\left.\frac{\partial q}{\partial t}\right|_{flush} = w_f \frac{\partial q_{br}}{\partial z}$$

$$\left.\frac{\partial S}{\partial t}\right|_{flush} = w_f \frac{\partial S_{br}}{\partial z}$$

These equations are discretized with an upwind advection scheme. The flushing Darcy flow, w_f, is given by

$$w_f = \frac{\overline{\Pi} \rho_w g \Delta h}{h \eta},$$

where $\overline{\Pi}$ is the harmonic mean of the ice layer permeabilities and Δh is the hydraulic head driving melt water through the sea ice. It is the difference in height between the top of the melt pond and sea level.

Basal boundary condition

In traditional Stefan problems the ice growth rate is calculated by determining the difference in heat flux on either side of the ice/ocean interface and equating this energy difference to the latent heat of new ice formed. Thus,

$$(1 - \phi_i) L_0 \rho_i \frac{\partial h}{\partial t} = K \left.\frac{\partial T}{\partial z}\right|_i - K_w \left.\frac{\partial T}{\partial z}\right|_w \tag{2.109}$$

where $(1 - \phi_i)$ is the solid fraction of new ice formed and the right hand is the difference in heat flux at the ice–ocean interface between the ice side and the ocean side of the interface. However, with mushy layers there is usually no discontinuity in solid fraction across the interface, so $\phi_i = 1$ and Equation (2.109) cannot be used explicitly. To circumvent this problem we set the interface solid fraction to be 0.15, a value that reproduces observations. ϕ_i is a namelist parameter (`phi_i_mushy` = 0.85). The basal ice temperature is set to the liquidus temperature T_f of the ocean surface salinity.

Tracer consistency

In order to ensure conservation of energy and salt content, the advection routines will occasionally limit changes to either enthalpy or bulk salinity. The mushy thermodynamics routine determines temperature from both enthalpy and bulk salinity. Since the limiting changes performed in the advection routine are not applied consistently (from a mushy physics point of view) to both enthalpy and bulk salinity, the resulting temperature may be changed to be greater than the limit allowed in the thermodynamics routines. If this situation is detected, the code corrects the enthalpy so the temperature is below the limiting value. The limiting value, `Tliquidus_max` can be specified in namelist. Conservation of energy is ensured by placing the excess energy in the ocean, and the code writes a warning (see *Error Messages and Aborts*) that this has occurred to the diagnostics file. This situation only occurs with the mushy thermodynamics, and it should only occur very infrequently and have a minimal effect on results. The addition of the heat to the ocean may reduce ice formation by a small amount afterwards.

2.7.5 Growth and melting

Melting at the top surface is given by

$$q\,\delta h = \begin{cases} (F_0 - F_{ct})\,\Delta t & \text{if } F_0 > F_{ct} \\ 0 & \text{otherwise} \end{cases} \tag{2.110}$$

where q is the enthalpy of the surface ice or snow layer[1] (recall that $q < 0$) and δh is the change in thickness. If the layer melts completely, the remaining flux is used to melt the layers beneath. Any energy left over when the ice and snow are gone is added to the ocean mixed layer. Ice cannot grow at the top surface due to conductive fluxes; however, snow–ice can form. New snowfall is added at the end of the thermodynamic time step.

Growth and melting at the bottom ice surface are governed by

$$q\,\delta h = (F_{cb} - F_{bot})\,\Delta t, \tag{2.111}$$

where F_{bot} is given by Equation (2.92) and F_{cb} is the conductive heat flux at the bottom surface:

$$F_{cb} = \frac{K_{i,N+1}}{\Delta h_i}(T_{iN} - T_f).$$

If ice is melting at the bottom surface, q in Equation (2.111) is the enthalpy of the bottom ice layer. If ice is growing, q is the enthalpy of new ice with temperature T_f and salinity S_{max} (`ktherm` = 1) or ocean surface salinity (`ktherm` = 2). This ice is added to the bottom layer.

In general, frazil ice formed in the ocean is added to the thinnest ice category. The new ice is grown in the open water area of the grid cell to a specified minimum thickness; if the open water area is nearly zero or if there is more new ice than will fit into the thinnest ice category, then the new ice is spread over the entire cell.

If `tr_fsd=true`, a floe size must be assigned to the new frazil ice. If spectral ocean surface wave forcing is provided (and set using the namelist option `wave_spec_type`), this will be used to calculate a tensile stress on new floes that determines their maximum possible size [62][53]. If no ocean surface wave forcing is provided, all floes are assumed to grow as pancakes, at the smallest possible floe size.

If `tr_fsd=true`, lateral growth at the edges of exisiting floes may also occur, calculated using the prognostic floe size distribution as described in [24] and [51]. The lateral growth that occurs is a portion of the total new ice growth, depending on the area of open water close to floe edges. Lateral growth modifies the ITD and the FSD.

If `tr_fsd=true`, floes may weld together thermodynamically during freezing conditions according to the probability that they overlap, assuming they are replaced randomly on the domain. Evolution of the FSD is described using a coagulation equation. The total number of floes that weld with another, per square meter, per unit time, in the case of a fully covered ice surface was estimated from observations in [52]. In its original model implementation, with 12 floe size categories, the tendency term for floe welding was divided by a constant equal to the area of the largest floe, (approx 2 km^2), with this choice made as the product of sensitivity studies to balance the climatological tendencies of wave fracture and welding. So that results do not vary as the number or range of floe size categories varies, we fix this scaling coefficient, c_weld.

If the latent heat flux is negative (i.e., latent heat is transferred from the ice to the atmosphere), snow or snow-free ice sublimates at the top surface. If the latent heat flux is positive, vapor from the atmosphere is deposited at the surface as snow or ice. The thickness change of the surface layer is given by

$$(\rho L_v - q)\delta h = F_l \Delta t, \tag{2.112}$$

where ρ is the density of the surface material (snow or ice), and $L_v = 2.501 \times 10^6$ J/kg is the latent heat of vaporization of liquid water at $0°C$. Note that ρL_v is nearly an order of magnitude larger than typical values of q. For positive latent heat fluxes, the deposited snow or ice is assumed to have the same enthalpy as the existing surface layer.

[1] The mushy thermodynamics option does not include the enthalpy associated with raising the meltwater temperature to in these calculations, unlike Bitz99, which does include it. This extra heat is returned to the ocean (or the atmosphere, in the case of evaporation) with the melt water.

After growth and melting, the various ice layers no longer have equal thicknesses. We therefore adjust the layer interfaces, conserving energy, so as to restore layers of equal thickness $\Delta h_i = h_i/N_i$. This is done by computing the overlap η_{km} of each new layer k with each old layer m:

$$\eta_{km} = \min(z_m, z_k) - \max(z_{m-1}, z_{k-1}),$$

where z_m and z_k are the vertical coordinates of the old and new layers, respectively. The enthalpies of the new layers are

$$q_k = \frac{1}{\Delta h_i} \sum_{m=1}^{N_i} \eta_{km} q_m.$$

If `tr_fsd=false`, lateral melting is accomplished by multiplying the state variables by $1 - r_{side}$, where r_{side} is the fraction of ice melted laterally [44][64], and adjusting the ice energy and fluxes as appropriate. We assume a floe diameter of 300 m.

If `tr_fsd=true`, lateral melting is accomplished using the [44] lateral heat flux, but applied to the ice using the prognostic floe size distribution as described in [24] and [51]. Lateral melt modifies the ITD and the FSD.

2.7.6 Snow-ice formation

At the end of the time step we check whether the snow is deep enough to lie partially below the surface of the ocean (freeboard). From Archimedes' principle, the base of the snow is at sea level when

$$\rho_i h_i + \rho_s h_s = \rho_w h_i.$$

Thus the snow base lies below sea level when

$$h^* \equiv h_s - \frac{(\rho_w - \rho_i) h_i}{\rho_s} > 0.$$

In this case, for `ktherm` = 1 (Bitz99) we raise the snow base to sea level by converting some snow to ice:

$$\begin{aligned} \delta h_s &= \frac{-\rho_i h^*}{\rho_w}, \\ \delta h_i &= \frac{\rho_s h^*}{\rho_w}. \end{aligned}$$

In rare cases this process can increase the ice thickness substantially. For this reason snow–ice conversions are postponed until after the remapping in thickness space (*Transport in thickness space*), which assumes that ice growth during a single time step is fairly small.

For `ktherm` = 2 (mushy), we model the snow–ice formation process as follows: If the ice surface is below sea level then we replace some snow with the same thickness of sea ice. The thickness change chosen is that which brings the ice surface to sea level. The new ice has a porosity of the snow, which is calculated as

$$\phi = 1 - \frac{\rho_s}{\rho_i}$$

where ρ_s is the density of snow and ρ_i is the density of fresh ice. The salinity of the brine occupying the above porosity within the new ice is taken as the sea surface salinity. Once the new ice is formed, the vertical ice and snow layers are regridded into equal thicknesses while conserving energy and salt.

2.8 Advanced snow physics

Once deposited, the character and distribution of snow on sea ice depend on re-transport (wind), melting/wetting, and metamorphism (chiefly producing low-conductivity depth hoar or snow-ice). Each of these processes affects the other, and they are crucial for the evolution of the sea ice pack [65]. In particular, Wind slab and depth hoar resist densification, and Wind slab may prevent snow from drifting after deposition. Snow drifts around ridges cover only 6% of the ice surface area and are about 30% deeper than other snow-covered areas, but they prevent seawater filled cracks around the ridges from freezing, with important biological consequences.

The standard model configuration includes a basic snow formulation describing the essential effects of snow on sea ice, such as its albedo, vertical conduction, and growth/melt processes. It also incorporates more detailed processes such as snow-ice formation due to flooding and snow infiltration by melt water, which may form melt ponds. Several potentially important processes are not included in the standard configuration, such as compaction and redistribution of snow by wind and their effects on the thermal balance and on effective roughness. Snow metamorphism due to temperature gradients and liquid water content also are not included.

Setting `tr_snow = .true.` activates advanced snow physics parameterizations that represent the following processes, each of which has its own namelist flag for flexible configuration:

1. Radiative effects of snow redistribution by wind with respect to ice topography, including snow loss to leads and snow compaction by wind
2. Coupling effects associated with snow saturation and pond formation.
3. Radiative effects of snow grain metamorphism (variable grain size)

Snow can be scoured from level ice, blowing into leads or piling up on ridges. The presence of liquid water in snow, such as rain or melt water, changes the surface albedo dramatically. It also alters the conductivity of the snow pack. These effects are associated mainly with the formation of depth hoar (change in grain size).

The standard model configuration assumes that the snow depth is uniform across each ice thickness category within a grid cell for the vertical thermodynamic calculation. However, there are separate radiation calculations for bare ice, snow-covered ice, and pond-covered ice; snow and ponds interact through snow saturation levels. Redistributing the snow alters these radiative calculations.

2.8.1 Snow redistribution

Because the thermodynamic schemes in CICE assume a uniform snow depth over each category, ignoring the fractions of level and deformed ice, effects of snow redistribution are included only via the delta-Eddington radiation scheme. The redistributed snow depth is used to determine the effective area of bare ice (for very small snow depths) and the effective area and depth of melt ponds over level ice. Once those areas are determined, the redistributed snow volume over them is known, from which the snow depth for the remaining snow-covered area can be computed and used for its radiation balance calculation.

Two basic approaches are available for snow redistribution by wind, `snwredist = bulk`, for which a user-defined parameter p (`snwlvlfac`) determines the ratio of snow on ridges to that on level ice, and `snwITDrdg`, in which snow can be compacted by the wind or eroded and redeposited on other thickness categories. For both, nonlocal redistribution of snow (i.e., between grid cells) is neglected, assuming that the difference between snow mass blowing into a grid cell and that blowing out is negligible, but snow can be blown into nearby leads and open water.

Bulk snow redistribution

[65] noted that on average during the SHEBA experiment, snow near ridged ice was 30% deeper than snow on undeformed ice. Using this rule of thumb, we can reduce the amount of snow on level ice in the model by reducing the snowfall rate over the sea ice and assuming the removed snow volume passes into the ocean through leads, instantaneously. This approach takes into account the area of open water available, as in the original code, by employing a precipitation flux in units of kg m $^{-2}$ s $^{-1}$, which accumulates snow only on the ice-covered area of the grid cell.

This approach affects the simulation in two ways: (1) the snow removed from the level ice area is deposited into leads, and (2) using the snow remaining on the level ice area to adjust the effective melt pond and bare ice areas. Case (1) affects both the radiative and thermodynamic calculations by reducing the total amount of snow on the ice. Case (2) affects the radiative calculation directly, by possibly exposing more bare ice or melt ponds, but it affects the thermodynamic (conduction) calculation only through the altered radiative absorption, since the snow is always assumed to be equally deep over both level and deformed ice for the thermodynamic calculation.

When `snwredist = bulk`, snow loss to leads is accomplished simply by reducing the volume of snowfall reaching the ice:

$$f_s' = f_s \left[a_{lvl} \left(\frac{p}{1+p} \right) \right],$$

where f_s is the snowfall rate, a_{lvl} is the average level-ice tracer value, and primed quantities represent their modified values.

Snow is redistributed between level and ridged ice within a single thickness category by solving a pair of equations for the modified level- and ridged-ice snow depths in terms of the original snow depth:

$$h'_{lvl} = \frac{1}{1+p(1-a_{lvl})} h_{lvl}$$

$$h'_{rdg} = \frac{1+p}{1+p(1-a_{lvl})} h_{lvl}.$$

In the shortwave module for level-ice ponds, we create a new variable h'_{lvl} (`hsnlvl`) for snow depth over the level ice, and replace `hsn` with `hsnlvl` for the snow infiltration calculation and for the calculation of snow depth over refrozen melt ponds.

Snow redistribution and compaction by wind

Following [34], when `snwredist = snwITDrdg` we parameterize the amount of snow lost into the ocean through leads or redistributed to other thickness categories by defining the redistribution function Φ for snow mass as the sum of an erosion rate Φ_E and a redeposition rate Φ_R for each category of thickness h_i:

$$\Phi_E = \left(\frac{\partial m}{\partial t} \right)_{erosion} = -\frac{\gamma}{\sigma_{ITD}} (V - V^*) \frac{\rho_{max} - \rho_s}{\rho_{max}}$$

where ρ_s and ρ_{max} are the effective snow density and the maximum snow density in the model, respectively. For now, we take ρ_s to be the wind-compacted snow density computed at the end of the snow model time step.

$\Phi_E \Delta t$ represents the maximum snow mass per unit area that may be suspended from each category, subject to the total mass (per unit area) available on each category.

Erosion begins when the instantaneous wind speed V exceeds the seasonal wind speed required to compact the snow to a density ρ_s, $V^* = (\rho_s - \beta)/\alpha$. σ_{ITD} is the standard deviation of the ice thicknesses from the thickness distribution g within the grid cell. γ is a tuning coefficient for the eroded mass, which [34] set to 10^{-5} kg m $^{-2}$. From [35], $\rho_s = 44.6V^* + 174$ kg m $^{-3}$ for seasonal mean wind speed V, i.e. $\alpha = 174$ kg m $^{-3}$ and $\beta = 44.6$ kg s m $^{-4}$.

In [34], the fraction of this suspended snow lost in leads is

$$f = (1 - a_i) \exp \left(\frac{-\sigma_{ITD}}{\sigma_{ref}} \right),$$

where the scale factor $\sigma_{ref} = 1$ m and a_i is the total ice area fraction within the grid cell. Thus, the snow mass that is redistribution on the ice (i.e., not lost in leads) is

$$\Phi_R \Delta t = a_i \left(1 - f\right) \Phi_E \Delta t.$$

We extend this approach by using the level and ridged ice thicknesses to compute the standard deviation of ice thickness across all categories. That is,

$$\sigma^2_{ITD} = \sum_{n=1}^{N} a_{in} a_{lvln} \left(h_{ilvln} - \sum_{k=1}^{N} a_{ik} h_{ik} \right)^2 + a_{in} a_{rdgn} \left(h_{irdgn} - \sum_{k=1}^{N} a_{ik} h_{ik} \right)^2 .$$

When considering snow over ridged and level ice for the redistribution, we reapportion the fraction of snow on level ice as $a_{slvl} = 1 - (1 + p) a_{rdg}$ and note that with the average expression

$$a_{slvl} = \frac{\sum_{n=1}^{N} a_{in} \left(a_{lvln} - p a_{rdgn} \right)}{\sum_{n=1}^{N} a_{in}}$$

a conservative redistribution of snow across thickness categories is (for each category n)

$$\Phi_R(n) \Delta t = a_i \left(1 - f\right) \left[a_{rdgn} \left(1 + p\right) + a_{slvl} \right] \Phi_E \Delta t,$$

where $p \le a_{lvln}/a_{rdgn}$.

The snow volume and energy state variables are updated in two steps, first for erosion of snow into suspension, then snow redeposition. When redepositing the snow, the snow energy is distributed among the snow layers affected by erosion, proportionally to the fraction of snow eroded. Finally, snow layer thicknesses are re-equalized, conserving snow energy. The fraction of suspended snow mass and energy lost in leads is added to the fresh water and heat fluxes for strict conservation.

High wind speeds compact the upper portion of a snow pack into “wind slab,” a dense and more conductive medium that resists further drifting. An effective snow density is computed based on wind speed, which is then used to limit snow erosion of denser snow.

[34] note that once snow is deposited, its density changes very little. During deposition, the density primarily falls into one of two types, wind slab for wind velocities greater than about 10 m/s, and loose snow for lighter winds. Their table 3 indicates densities for a variety of snow types. “Hard slab,” deposited at V = 13 m/s, has a density of ρ_s = 403 kg m $^{-3}$ and “soft slab” is ρ_s = 321 kg m $^{-3}$, deposited at V = 10 m/s. Linearly interpolating between these values, we have $\rho_s = 27.3V + 47.7$. The slope is an adjustable namelist parameter, `drhosdwind`. For simplicity, we assign a minimum snow density of ρ_s^{min} = 100 kg m $^{-3}$ s (`rhosmin`) and add to it the gradient associated with wind speed from [34] for wind speeds greater than 10 m/s: $\rho_s^{new} = \rho_s^{min} + 27.3 \max \left(V - 10, 0\right)$. The minimum wind speed to compact snow `windmin` is adjustable, and the maximum snow density is also a namelist parameter, `rhosmax`. This density is merged with preexisting layer densities only if new snow falls. The thickness of the wind slab is the larger of the depth of newly fallen snow or the thickness of snow redeposited by the wind. Following [65], density does not evolve further, other than by transport, unless additional snow falls at high enough wind speeds to compact the snow.

2.8.2 Ice and liquid water mass in snow

The advanced snow physics option calculates ice and liquid water mass and effective snow grain radius, enabling them to interact with the radiation calculation. The mass of ice and liquid water in snow are implemented as tracers on snow volume layers and used for the snow grain metamorphism. Together with snow volume, they also can be used to determine effective snow density as $\rho_s^{eff} = (m_{ice} + m_{liq})/h_s$. Note that $m_{ice} + m_{liq}$ is the snow water equivalent (kg/m ^{2}).

Sources of m_{ice} are snowfall, condensation, and freezing of liquid water within the snowpack; sinks are sublimation and melting. All of the sources and sinks of m_{ice} are already computed in the code except for freezing of liquid water within the snow pack.

Sources of m_{liq} are rain and snow melt; freezing of liquid water within the snowpack and runoff are sinks. Runoff and meltwater entering a snow layer (i.e., runoff from the layer above) are associated with vertical flow through the snow column. As in [47], when the liquid water within a snow layer exceeds the layer's holding capacity, the excess water is added to the underlying layer, limited by the effective porosity of the layer. When `use_smliq_pnd` is true, the excess water is supplied to the melt pond parameterization, which puts a fraction of it into the pond volume and allows the rest to run off into the ocean.

The snow mass fractions of precipitation and old ice are saved for metamorphosing the snow grain radius.

Except for the topo melt pond scheme, melt water and heat in ponds (which may be hidden within a partially saturated snow pack) are "virtual" in the sense that they are provided to the ocean model component immediately upon melting, even though the effects of the liquid water continue to be tracked as if it were retained on the ice. Retaining that water and heat in the sea ice component alters the timing, location and magnitude of fresh water runoff events into the ocean. All melt pond schemes include the meltwater effects, regardless of whether the liquid water is virtual. The advanced snow physics option allows the liquid water calculated by the snow metamorphism scheme to be used for melt pond calculations, replacing the snow melt and rainfall terms.

2.8.3 Metamorphosis of snow grains

When `snwgrain = .true.`, dynamic, effective snow radius, a snow volume tracer, evolves analytically as a function of snow temperature, temperature gradient, and density for radiative calculations using the delta-Eddington radiation scheme. Wet metamorphism changes both density (through volume change) and effective grain size; here we only consider changes in grain radius. In the formation of depth hoar, dry snow kinetic metamorphism (TG metamorphism) also increases the snow grain radius.

The tracers m_{liq} and m_{ice} characterize the snow in each snow layer, for each ice category and horizontal grid cell. The model's meltpond volume covers a fraction of the grid cell and represents liquid in excess of m_{liq}. The radiative effects of snow grain radius in the fraction of ice covered by pond volume are only calculated when the pond volume has not yet saturated the snow pack; otherwise, delta-Eddington transfer uses meltpond properties. Therefore, modelled changes in snow grain radii from metamorphism are designed specifically for the fraction without exposed (i.e. effective) melt ponds.

Following [47], the new snow grain radius is computed as a weighted function of existing and new (freshly fallen, `rsnw_fall`) snow grain radii, using parameters from a look-up table that depends on snow temperature, temperature gradient and (effective) density. The maximum snow radius is a namelist option, `rsnw_tmax`.

2.9 Biogeochemistry

2.9.1 Aerosols

Basic Aerosols

Aerosols may be deposited on the ice and gradually work their way through it until the ice melts and they are passed into the ocean. They are defined as ice and snow volume tracers (Eq. 15 and 16 in CICE.v5 documentation), with the snow and ice each having two tracers for each aerosol species, one in the surface scattering layer (delta-Eddington SSL) and one in the snow or ice interior below the SSL.

Rather than updating aerosols for each change to ice/snow thickness due to evaporation, melting, snow-ice formation, etc., during the thermodynamics calculation, these changes are deduced from the diagnostic variables (melts, meltb, snoice, etc) in **icepack_aerosol.F90**. Three processes change the volume of ice or snow but do not change the total amount of aerosol, thus causing the aerosol concentration (the value of the tracer itself) to increase: evaporation, snow deposition and basal ice growth. Basal and lateral melting remove all aerosols in the melted portion. Surface ice and snow melt leave a significant fraction of the aerosols behind, but they do "scavenge" a fraction of them given by the parameter `kscav` = [0.03, 0.2, 0.02, 0.02, 0.01, 0.01] (only the first 3 are used in CESM, for their 3 aerosol species). Scavenging also applies to snow-ice formation. When sea ice ridges, a fraction of the snow on the ridging ice is thrown into the ocean, and any aerosols in that fraction are also lost to the ocean.

As upper SSL or interior layers disappear from the snow or ice, aerosols are transferred to the next lower layer, or into the ocean when no ice remains. The atmospheric flux `faero_atm` contains the rates of aerosol deposition for each species, while `faero_ocn` has the rate at which the aerosols are transferred to the ocean.

The aerosol tracer flag `tr_aero` must be set to true in **icepack_in**, and the number of aerosol species is set in **icepack.settings**; CESM uses 3.

Z-Aerosols

An alternate scheme for aerosols in sea ice is available using the brine motion based transport scheme of the biogeochemical tracers. All vertically resolved biogeochemical tracers (z-tracers), including aerosols, have the potential to be atmospherically deposited onto the snow or ice, scavenged during snow melt, and passed into the brine. The mobile fraction (discussed in *Mobile and stationary phases*) is then transported via brine drainage processes (Eq. (2.129)) while a stationary fraction (discussed in *Mobile and stationary phases*) adheres to the ice crystals. Snow deposition and the process of scavenging aerosols during snow melt is consistent with the basic aerosol scheme, though parameters have been generalized to accomodate potential atmospheric deposition for all z-tracers. For an example, see the scavenging parameter `kscavz` for z-tracers defined in **icepack_zbgc_shared.F90**.

Within the snow, z-tracers are defined as concentrations in the snow surface layer (h_{ssl}) and the snow interior ($h_s - h_{ssl}$). The total snow content of z-tracers per ice area per grid cell area, C_{snow} is

$$C_{snow} = C_{ssl} h_{ssl} + C_{sint}(h_s - h_{ssl})$$

One major difference in how the two schemes model snow aerosol transport is that the fraction scavenged from snow melt in the z-tracer scheme is not immediately fluxed into the ocean, but rather, enters the ice as a source of low salinity but potentially tracer-rich brine. The snow melt source is included as a surface flux condition in **icepack_algae.F90**.

All the z-aerosols are nonreactive with the exception of the dust aerosols. We assume that a small fraction of the dust flux into the ice has soluble iron (`dustFe_sol` in **icepack_in**) and so is passed to the dissolved iron tracer. The remaining dust passes through the ice without reactions.

To use z-aerosols, `tr_zaero` must be set to true in **icepack_in**, and the number of z-aerosol species is set in **icepack.settings**, `TRZAERO`. Note, the basic tracers `tr_aero` must be false and `NTRAERO` in **icepack.settings** should be 0. In addition, z-tracers and the brine height tracer must also be active. These are set in **icepack_in** with `tr_brine` and `z_tracer` set to true. In addition, to turn on the radiative coupling between the aerosols and the Delta-Eddington radiative scheme, `shortwave` must equal 'dEdd' and `dEdd_algae` must be true in **icepack_in**.

2.9.2 Water Isotope

Water isotopes may be deposited on the ice from above or below, and gradually work their way through it until the ice melts and they are passed into the ocean. They are defined as ice and snow volume tracers (Eq. 15 and 16 in CICE.v5 documentation), with the snow and ice each having one tracer for each water isotope species.

Rather than updating water isotopes for each change to ice/snow thickness due to evaporation, melting, snow-ice formation, etc., during the thermodynamics calculation, these changes are deduced from the diagnostic variables (melts, meltb, snoice, etc) in **icepack_isotope.F90**. The water isotopes follow the "real" water in the sense that all of the mass budget changes that impact fresh water equally affect the water isotopes. The sources of water isotopes are precipitation (snow accumulation), condensation, and sea ice growth, including both frazil and congelation formation that take up water isotopes from the ocean. The sinks are evaporation and melting snow and sea ice. Isotopic fractionation occurs for vapor condensation and new sea ice growth, in which H2_16O (regular water, ${H_2}O$) is treated differently than H2_18O and HDO.

More information can be found in [7].

2.9.3 Brine height

The brine height, h_b, is the distance from the ice-ocean interface to the brine surface. When `tr_brine` is set true in **icepack_in** and `TRBRI` is set equal to 1 in **icepack.settings**, the brine surface can move relative to the ice surface. Physically, this occurs when the ice is permeable and there is a nonzero pressure head: the difference between the brine height and the equilibrium sea surface. Brine height motion is computed in **icepack_brine.F90** from thermodynamic variables and the ice microstructural state deduced from internal bulk salinities and temperature. This tracer is required for the transport of vertically resolved biogeochemical tracers.

Vertical transport processes are, generally, a result of the brine motion. Therefore the vertical transport equations for biogeochemical tracers will be defined only where brine is present. This region, from the ice-ocean interface to the brine height, defines the domain of the vertical bio-grid. The resolution of the bio-grid is specified in **icepack.settings** by setting the variable `NBGCLYR`. A detailed description of the bio-grid is given in section *Grid and boundary conditions*. The ice microstructural state, determined in **icepack_brine.F90**, is computed from sea ice salinities and temperatures linearly interpolated to the bio-grid. When $h_b > h_i$, the upper surface brine is assumed to have the same temperature as the ice surface.

Brine height is transported horizontally as the fraction $f_{bri} = h_b/h_i$, a volume conserved tracer. Note that unlike the sea ice porosity, brine height fraction may be greater than 1 when $h_b > h_i$.

Changes to h_b occur from ice and snow melt, ice bottom boundary changes, and from pressure adjustments. The computation of h_b at $t + \Delta t$ is a two step process. First, h_b is updated from changes in ice and snow thickness, ie.

$$h_b' = h_b(t) + \Delta h_b|_{h_i,h_s}. \tag{2.113}$$

Second, pressure driven adjustments arising from meltwater flushing and snow loading are applied to h_b'. Brine flow due to pressure forces are governed by Darcy's equation

$$w = -\frac{\Pi^* \bar{\rho} g}{\mu} \frac{h_p}{h_i}. \tag{2.114}$$

The vertical component of the net permeability tensor Π^* is computed as

$$\Pi^* = \left(\frac{1}{h} \sum_{i=1}^{N} \frac{\Delta z_i}{\Pi_i} \right)^{-1} \tag{2.115}$$

where the sea ice is composed of N vertical layers with ith layer thickness Δz_i and permeability Π_i. The average sea ice density is $\bar{\rho}$ specified in **icepack_zbgc_shared.F90**. The hydraulic head is $h_p = h_b - h_{sl}$ where h_{sl} is the sea level given by

$$h_{sl} = \frac{\bar{\rho}}{\rho_w} h_i + \frac{\rho_s}{\rho_w} h_s. \tag{2.116}$$

Assuming constant h_i and h_s during Darcy flow, the rate of change of h_b is

$$\frac{\partial h_b}{\partial t} = -w h_p \tag{2.117}$$

where $w_o = \Pi^* \bar{\rho} g / (h_i \mu \phi_{top})$ and ϕ_{top} is the upper surface porosity. When the Darcy flow is downward into the ice ($w_o < 0$), then ϕ_{top} equals the sea ice porosity in the uppermost layer. However, when the flow is upwards into the snow, then ϕ_{top} equals the snow porosity phi_snow specified in **icepack_in**. If a negative number is specified for phi_snow, then the default value is used: phi_snow $= 1 - \rho_s/\rho_w$.

Since h_{sl} remains relatively unchanged during Darcy flow, (2.117) has the approximate solution

$$h_b(t + \Delta t) \approx h_{sl}(t + \Delta t) + [h_b' - h_{sl}(t + \Delta t)] \exp\{-w \Delta t\}. \tag{2.118}$$

The contribution $\Delta h_b|_{h_i,h_s}$ arises from snow and ice melt and bottom ice changes. Since the ice and brine bottom boundaries coincide, changes in the ice bottom from growth or melt, $(\Delta h_i)_{bot}$, equal the bottom brine boundary changes. The surface contribution from ice and snow melt, however, is opposite in sign. The ice contribution is as follows. If $h_i > h_b$ and the ice surface is melting, ie. $(\Delta h_i)_{top} < 0$), then meltwater increases the brine height:

$$(\Delta h_b)_{top} = \frac{\rho_i}{\rho_o} \cdot \begin{cases} -(\Delta h_i)_{top} & \text{if } |(\Delta h_i)_{top}| < h_i - h_b \\ h_i - h_b & \text{otherwise.} \end{cases} \tag{2.119}$$

For snow melt ($\Delta h_s < 0$), it is assumed that all snow meltwater contributes a source of surface brine. The total change from snow melt and ice thickness changes is

$$\Delta h_b|_{h_i,h_s} = (\Delta h_b)_{top} - (\Delta h_i)_{bot} - \frac{\rho_s}{\rho_o} \Delta h_s. \tag{2.120}$$

The above brine height calculation is used only when h_i and h_b exceed a minimum thickness, thinS, specified in **icepack_zbgc_shared.F90**. Otherwise

$$h_b(t + \Delta t) = h_b(t) + \Delta h_i \tag{2.121}$$

provided that $|h_{sl} - h_b| \leq 0.001$. This formulation ensures small Darcy velocities when h_b first exceeds thinS.

Both the volume fraction f_{bri} and the area-weighted brine height h_b are available for output.

$$\frac{\sum f_{bri} v_i}{\sum v_i}, \tag{2.122}$$

while `hbri` is comparable to hi (h_i)

$$\frac{\sum f_{bri} h_i a_i}{\sum a_i}, \tag{2.123}$$

where the sums are taken over thickness categories.

2.9.4 Sea ice ecosystem

There are two options for modeling biogeochemistry in sea ice: 1) a skeletal layer or bottom layer model that assumes biology and biological molecules are restricted to a single layer at the base of the sea ice; and 2) a vertically resolved model (zbgc) that allows for biogeochemical processes throughout the ice column. The two models may be run with the same suite of biogeochemical tracers and use the same module **algal_dyn** in **icepack_algae.F90** to determine the biochemical reaction terms for the tracers at each vertical grid level. In the case of the skeletal-layer model this is a single layer, while for zbgc there are NBGCLYR+1 vertical layers. The primary difference between the two schemes is in the vertical transport assumptions for each biogeochemical tracer. This includes the parameterizations of fluxes between ocean and ice.

In order to run with the skeletal-layer model, the code must be built with the following options in **icepack.settings**:

```
setenv TRBGCS 1   # set to 1 for skeletal layer tracers
setenv TRBGCZ 0   # set to 1 for zbgc tracers
```

For zbgc with 8 vertical layers:

```
setenv TRBRI  1   # set to 1 for brine height tracer
setenv TRBGCS 0   # set to 1 for skeletal layer tracers
setenv TRBGCZ 1   # set to 1 for zbgc tracers
setenv NBGCLYR 7  # number of zbgc layers
```

There are also environmental variables in **icepack.settings** that, in part, specify the complexity of the ecosystem and are used for both zbgc and the skeletal-layer model. These are 1) TRALG, the number of algal species; 2) TRDOC, the number of dissolved organic carbon groups, 3) TRDIC, the number of dissolved inorganic carbon groups (this is currently not yet implemented and should be set to 0); 4) TRDON, the number of dissolved organic nitrogen groups, 5) TRFEP, the number of particulate iron groups; and 6) TRFED, the number of dissolved iron groups. The current version of **algal_dyn** biochemistry has parameters for up to 3 algal species (diatoms, small phytoplankton and *Phaeocystis* sp, respectively), 2 DOC tracers (polysaccharids and lipids, respectively), 0 DIC tracers, 1 DON tracer (proteins/amino acids), 1 particulate iron tracer and 1 dissolved iron tracer. Note, for tracers with multiple species/groups, the order is important. For example, specifying TRALG = 1 will compute reaction terms using parameters specific to ice diatoms. However, many of these parameters can be modified in **icepack_in**.

The complexity of the algal ecosystem must be specified in both **icepack.settings** during the build and in the namelist, **icepack_in**. The procedure is equivalent for both the skeletal-layer model and zbgc. The namelist specification is described in detail in section *Vertical BGC (''zbgc'')*

Biogeochemical upper ocean concentrations are initialized in the subroutine **icepack_init_ocean_conc** in **icepack_zbgc.F90** unless coupled to the ocean biogeochemistry. Silicate and nitrate may be read from a file. This option is specified in the namelist by setting the variables bgc_data_type to ISPOL or NICE. The location of forcing files is specified in data_dir and the filename is also in namelist, bgc_data_file.

Skeletal Layer BGC

In the skeletal layer model, biogeochemical processing is modelled as a single layer of reactive tracers attached to the sea ice bottom. Optional settings are available via the *zbgc_nml* namelist in **icepack_in**. In particular, skl_bgc must be true and z_tracers and solve_zbgc must both be false.

Skeletal tracers T_b are ice area conserved and follow the horizontal transport Equation (2.25). For each horizontal grid point, local biogeochemical tracer equations are solved in **icepack_algae.F90**. There are two types of ice-ocean tracer flux formulations: 1) 'Jin2006' modeled after the growth rate dependent piston velocity and 2) 'constant' modeled after a constant piston velocity. The formulation is specified in **icepack_in** by setting bgc_flux_type equal to 'Jin2006' or 'constant'.

In addition to horizontal advection and transport among thickness categories, biogeochemical tracers (T_b where $b = 1, \ldots, N_b$) satisfy a set of local coupled equations of the form

$$\frac{dT_b}{dt} = w_b \frac{\Delta T_b}{\Delta z} + R_b(T_j : j = 1, \ldots, N_b) \tag{2.124}$$

where R_b represents the nonlinear biochemical reaction terms (described in section *Reaction Equations*) and Δz is a length scale representing the molecular sublayer of the ice-ocean interface. Its value is absorbed in the piston velocity parameters. The piston velocity w_b depends on the particular tracer and the flux formulation.

For 'Jin2006', the piston velocity is a function of ice growth and melt rates. All tracers (algae included) flux with the same piston velocity during ice growth, $dh/dt > 0$:

$$w_b = \quad -p_g \left| m_1 + m_2 \frac{dh}{dt} - m_3 \left(\frac{dh}{dt} \right)^2 \right| \tag{2.125}$$

with parameters m_1, m_2, m_3 and p_g defined in **skl_biogeochemistry** in **icepack_algae.F90**. For ice melt, $dh/dt < 0$, all tracers with the exception of ice algae flux with

$$w_b = \quad p_m \left| m_2 \frac{dh}{dt} - m_3 \left(\frac{dh}{dt} \right)^2 \right| \tag{2.126}$$

with p_m defined in **skl_biogeochemistry**. The 'Jin2006' formulation also requires that for both expressions, $|w_b| \le 0.9 h_{sk}/\Delta t$. The concentration difference at the ice-ocean boundary for each tracer, ΔT_b, depends on the sign of w_b. For growing ice, $w_b < 0$, $\Delta T_b = T_b/h_{sk} - T_{io}$, where T_{io} is the ocean concentration of tracer i. For melting ice, $w_b > 0$, $\Delta T_b = T_b/h_{sk}$.

In 'Jin2006', the algal tracer (N_a) responds to ice melt in the same manner as the other tracers (2.126). However, this is not the case for ice growth. Unlike dissolved nutrients, algae are able to cling to the ice matrix and resist expulsion during desalination. For this reason, algal tracers do not flux between ice and ocean during ice growth unless the ice algal brine concentration is less than the ocean algal concentration (N_o). Then the ocean seeds the sea ice concentration according to

$$w_b \frac{\Delta N_a}{\Delta z} = \frac{N_o h_{sk}/\phi_{sk} - N_a}{\Delta t} \tag{2.127}$$

The 'constant' formulation uses a fixed piston velocity (PVc) for positive ice growth rates for all tracers except N_a. As in 'Jin2006', congelation ice growth seeds the sea ice algal population according to (2.127) when $N_a < N_o h_{sk}/\phi_{sk}$. For bottom ice melt, all tracers follow the prescription

$$w_b \frac{\Delta T_b}{\Delta z} = \begin{cases} T_b |dh_i/dt| / h_{sk} & \text{if } |dh_i/dt| \Delta t / h_{sk} < 1 \\ T_b/\Delta t & \text{otherwise.} \end{cases} \tag{2.128}$$

A detailed description of the biogeochemistry reaction terms is given in section *Reaction Equations*.

Vertical BGC (''zbgc")

In order to solve for the vertically resolved biogeochemistry, several flags in **icepack_in** must be true: a) `tr_brine`, b) `z_tracers`, and c) `solve_zbgc`.

- `tr_brine` = true turns on the dynamic brine height tracer, h_b, which defines the vertical domain of the biogeochemical tracers. z-Tracer horizontal transport is conserved on ice volume×brine height fraction.
- `z_tracers` = true indicates use of vertically resolved biogeochemical and z-aerosol tracers. This flag alone turns on the vertical transport scheme but not the biochemistry.
- `solve_zbgc` = true turns on the biochemistry for the vertically resolved tracers and automatically turns on the algal nitrogen tracer flag tr_bgc_N. If false, `tr_bgc_N` is set false and any other biogeochemical tracers in use are transported as passive tracers. This is appropriate for the black carbon and dust aerosols specified by `tr_zaero` true.

With the above flags, the default biochemistry is a simple algal-nitrate system: `tr_bgc_N` and `tr_bgc_Nit` are true. Options exist in **icepack_in** to use a more complicated ecosystem which includes up to three algal classes, two DOC groups, one DON pool, limitation by nitrate, silicate and dissolved iron, sulfur chemistry plus refractory humic material.

The **icepack_in** namelist options are described in the *Tables of Namelist Options*.

Vertically resolved z-tracers are brine- volume conserved and thus depend on both the ice volume and the brine height fraction tracer ($v_{in} f_b$). These tracers follow the conservation equations for multiply dependent tracers (see, for example Equation (2.70) where a_{pnd} is a tracer on $a_{lvl} a_i$)

The following sections describe the vertical transport equation for mobile tracers, the partitioning of tracers into mobile and stationary fractions and the biochemical reaction equations. The vertical bio-grid is described in the *Grid and boundary conditions* section.

Mobile and stationary phases

Purely mobile tracers are tracers which move with the brine and thus, in the absence of biochemical reactions, evolve like salinity. For vertical tracer transport of purely mobile tracers, the flux conserved quantity is the bulk tracer concentration multiplied by the ice thickness, i.e. $C = h\phi[c]$, where h is the ice thickness, ϕ is the porosity, and $[c]$ is the tracer concentration in the brine. ϕ, $[c]$ and C are defined on the interface bio grid (igrid):

$$\text{igrid}(k) = \Delta(k-1) \quad \text{for } k = 1 : n_b + 1 \text{ and } \Delta = 1/n_b.$$

The biogeochemical module solves the following equation:

$$\frac{\partial C}{\partial t} = \frac{\partial}{\partial x}\left\{\left(\frac{v}{h} + \frac{w_f}{h\phi} - \frac{\tilde{D}}{h^2\phi^2}\frac{\partial \phi}{\partial x}\right) C + \frac{\tilde{D}}{h^2\phi}\frac{\partial C}{\partial x}\right\} + h\phi R([c]) \tag{2.129}$$

where $D_{in} = \tilde{D}/h^2 = (D + \phi D_m)/h^2$ and $R([c])$ is the nonlinear biogeochemical interaction term (see [29]).

The solution to (2.129) is flux-corrected and positive definite. This is accomplished using a finite element Galerkin discretization. Details are in *Flux-corrected, positive definite transport scheme*.

In addition to purely mobile tracers, some tracers may also adsorb or otherwise adhere to the ice crystals. These tracers exist in both the mobile and stationary phases. In this case, their total brine concentration is a sum $c_m + c_s$ where c_m is the mobile fraction transported by equation (2.129) and c_s is fixed vertically in the ice matrix. The algae are an exception, however. We assume that algae in the stationary phase resist brine motion, but rather than being fixed vertically, these tracers maintain their relative position in the ice. Algae that adhere to the ice interior (bottom, surface), remain in the ice interior (bottom, surface) until release to the mobile phase.

In order to model the transfer between these fractions, we assume that tracers adhere (are retained) to the crystals with a time-constant of τ_{ret}, and release with a time constant τ_{rel}, i.e.

$$\begin{aligned}\frac{\partial c_m}{\partial t} &= -\frac{c_m}{\tau_{ret}} + \frac{c_s}{\tau_{rel}} \\ \frac{\partial c_s}{\partial t} &= \frac{c_m}{\tau_{ret}} - \frac{c_s}{\tau_{rel}}\end{aligned}$$

We use the exponential form of these equations:

$$\begin{aligned}c_m^{t+dt} &= c_m^t \exp\left(-\frac{dt}{\tau_{ret}}\right) + c_s^t\left(1 - \exp\left[-\frac{dt}{\tau_{rel}}\right]\right) \\ c_s^{t+dt} &= c_s^t \exp\left(-\frac{dt}{\tau_{rel}}\right) + c_m^t\left(1 - \exp\left[-\frac{dt}{\tau_{ret}}\right]\right)\end{aligned}$$

The time constants are functions of the ice growth and melt rates (dh/dt). All tracers except algal nitrogen diatoms follow the simple case: when $dh/dt \geq 0$, then $\tau_{rel} \to \infty$ and τ_{ret} is finite. For $dh/dt < 0$, then $\tau_{ret} \to \infty$ and τ_{rel} is finite. In other words, ice growth promotes transitions to the stationary phase and ice melt enables transitions to the mobile phase.

The exception is the diatom pool. We assume that diatoms, the first algal nitrogen group, can actively maintain their relative position within the ice, i.e. bottom (interior, upper) algae remain in the bottom (interior, upper) ice, unless melt rates exceed a threshold. The namelist parameter `algal_vel` sets this threshold.

The variable `bgc_tracer_type` determines the mobile to stationary transition timescales for each z-tracer. It is multi-dimensional with a value for each z-tracer. For `bgc_tracer_type``(k) equal to -1, the kth tracer remains solely in the mobile phase. For ``bgc_tracer_type` equal to 1, the tracer has maximal rates in the retention phase and minimal in the release. For `bgc_tracer_type` equal to 0, the tracer has maximal rates in the release phase and minimal in the retention. Finally for `bgc_tracer_type` equal to 0.5, minimum timescales are used for both transitions. Table *Types of Mobile and Stationary Transitions* summarizes the transition types. The tracer types are: `algaltype_diatoms`, `algaltype_sp` (small plankton), `algaltype_phaeo` (*phaeocystis*), `nitratetype`, `ammoniumtype`, `silicatetype`, `dmspptype`, `dmspdtype`, `humtype`, `doctype_s` (saccharids), `doctype_l` (lipids), `dontype_protein`, `fedtype_1`, `feptype_1`, `zaerotype_bc1` (black carbon class 1), `zaerotype_bc2` (black carbon class 2), and four dust classes, `zaerotype_dustj`, where j takes values 1 to 4. These may be modified to increase or decrease retention. Another option is to alter the minimum `tau_min` and maximum `tau_max` timescales which would impact all the z-tracers.

Table 3: *Types of Mobile and Stationary Transitions*

bgc_tracer_type	τ_{ret}	τ_{rel}	Description
-1.0	∞	0	entirely in the mobile phase
0.0	min	max	retention dominated
1.0	max	min	release dominated
0.5	min	min	equal but rapid exchange
2.0	max	max	equal but slow exchange

The fraction of a given tracer in the mobile phase is independent of ice depth and stored in the tracer variable zbgc_frac. The horizontal transport of this tracer is conserved on brine volume and so is dependent on two tracers: brine height fraction (f_b) and ice volume (v_{in}). The conservation equations are given by

$$\frac{\partial}{\partial t}(f_b v_{in}) + \nabla \cdot (f_b v_{in} \mathbf{u}) = 0.$$

The tracer, `zbgc_frac`, is initialized to 1 during new ice formation, because all z-tracers are initially in the purely mobile phase. Similarly, as the ice melts, z-tracers return to the mobile phase. Very large release timescales will prevent this transition and could result in an unphysically large accumulation during the melt season.

Flux-corrected, positive definite transport scheme

Numerical solution of the vertical tracer transport equation is accomplished using the finite element Galerkin discretization. Multiply (2.129) by "w" and integrate by parts

$$\begin{aligned} & \int_h \left[w\frac{\partial C}{\partial t} - \frac{\partial w}{\partial x}\left(-\left[\frac{v}{h} + \frac{w_f}{h\phi}\right] C + \frac{D_{in}}{\phi^2}\frac{\partial \phi}{\partial x} C - \frac{D_{in}}{\phi}\frac{\partial C}{\partial x}\right)\right] dx \\ + \quad & w\left(-\left[\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi}\right] C + \frac{D_{in}}{\phi^2}\frac{\partial \phi}{\partial x} C - \frac{D_{in}}{\phi}\frac{\partial C}{\partial x}\right)\Bigg|_{bottom} + w\left[\frac{1}{h}\frac{dh_t}{dt} + \frac{w_f}{h\phi}\right] C|_{top} = 0 \end{aligned}$$

The bottom boundary condition indicated by $|_{bottom}$ satisfies

$$\begin{aligned} -w\left(-\left[\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi}\right] C + \frac{D_{in}}{\phi^2}\frac{\partial \phi}{\partial x} C - \frac{D_{in}}{\phi}\frac{\partial C}{\partial x}\right)\Bigg|_{bottom} &= \\ w\left[\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi_{N+1}}\right](C_{N+2} \text{ or } C_{N+1}) - w\frac{D_{in}}{\phi_{N+1}(\Delta h + g_o)}(C_{N+1} - C_{N+2}) \end{aligned}$$

where $C_{N+2} = h\phi_{N+1}[c]_{ocean}$ and $w = 1$ at the bottom boundary and the top. The component C_{N+2} or C_{N+1} depends on the sign of the advection boundary term. If $dh_b + w_f/\phi > 0$ then use C_{N+2} otherwise C_{N+1}.

Define basis functions as linear piecewise, with two nodes (boundary nodes) in each element. Then for $i > 1$ and $i < N+1$

$$w_i(x) = \begin{cases} 0 & x < x_{i-1} \\ (x - x_{i-1})/\Delta & x_{i-1} < x \le x_i \\ 1 - (x - x_i)/\Delta & x_i \le x < x_{i+1} \\ 0, & x \ge x_{i+1} \end{cases}$$

For $i = 1$

$$w_1(x) = \begin{cases} 1 - x/\Delta & x < x_2 \\ 0, & x \ge x_2 \end{cases}$$

and $i = N+1$

$$w_{N+1}(x) = \begin{cases} 0, & x < x_N \\ (x - x_N)/\Delta & x \ge x_N \end{cases}$$

Now assume a form

$$C_h = \sum_j^{N+1} c_j w_j$$

Then

$$\begin{aligned} \int_h C_h dx = & \quad c_1 \int_0^{x_2} \left(1 - \frac{x}{\Delta}\right) dx + c_{N+1} \int_{x_N}^{x_{N+1}} \frac{x - x_N}{\Delta} dx \\ + & \quad \sum_{j=2}^{N} c_j \left\{ \int_{j-1}^{j} \frac{x - x_{j-1}}{\Delta} dx + \int_j^{j+1} \left[1 - \frac{(x - x_j)}{\Delta}\right] dx \right\} \\ = & \quad \Delta \left[\frac{c_1}{2} + \frac{c_{N+1}}{2} + \sum_{j=2}^{N} c_j \right] \end{aligned}$$

Now this approximate solution form is substituted into the variational equation with $w = w_h \in \{w_j\}$

$$\begin{aligned} 0 = & \quad \int_h \left[w_h \frac{\partial C_h}{\partial t} - \frac{\partial w_h}{\partial x} \left(\left[-\frac{v}{h} - \frac{w_f}{h\phi} + \frac{D_{in}}{\phi^2} \frac{\partial \phi}{\partial x} \right] C_h - \frac{D_{in}}{\phi} \frac{\partial C_h}{\partial x} \right) \right] dx \\ + & \quad w_h \left(- \left[\frac{1}{h} \frac{dh_b}{dt} + \frac{w_f}{h\phi} \right] C_h + \frac{D_{in}}{\phi^2} \frac{\partial \phi}{\partial x} C - \frac{D_{in}}{\phi} \frac{\partial C_h}{\partial x} \right) \Big|_{bottom} + w_h \left[\frac{1}{h} \frac{dh_t}{dt} + \frac{w_f}{h\phi} \right] C_h|_{top} \end{aligned}$$

The result is a linear matrix equation

$$M_{jk} \frac{\partial C_k(t)}{\partial t} = [K_{jk} + S_{jk}] C_k(t) + q_{in}$$

where

$$\begin{aligned} M_{jk} = & \quad \int_h w_j(x) w_k(x) dx \\ K_{jk} = & \quad \left[-\frac{v}{h} - \frac{w_f}{h\phi} + \frac{D_{in}}{\phi^2} \frac{\partial \phi}{\partial x} \right] \int_h \frac{\partial w_j}{\partial x} w_k dx \\ - & \quad w_j \left(- \left[\frac{v}{h} + \frac{w_f}{h\phi_k} \right] w_k + \frac{D_{in}}{\phi^2} \frac{\partial \phi_k}{\partial x} w_k - \frac{D_{in}}{\phi_k} \frac{\partial w_k}{\partial x} \right) \Big|_{bot} \\ = & \quad -V_k \int_h \frac{\partial w_j}{\partial x} w_k dx - w_j \left(-V_k w_k - \frac{D_{in}}{\phi_k} \frac{\partial w_k}{\partial x} \right) \Big|_{bot} \\ S_{jk} = & \quad -\frac{D_{in}}{\phi_k} \int_h \frac{\partial w_j}{\partial x} \cdot \frac{\partial w_k}{\partial x} dx \\ q_{in} = & \quad -V C_t w_j(x)|_t \end{aligned}$$

and $C_{N+2} = h\phi_{N+1}[c]_{ocean}$.

For the top condition q_{in} is applied to the upper value C_2 when $VC_t < 0$, i.e. q_{in} is a source.

Compute the M_{jk} integrals:

$$\begin{aligned}
M_{jj} = & \int_{x_{j-1}}^{x_j} \frac{(x-x_{j-1})^2}{\Delta^2} dx + \int_{x_j}^{x_{j+1}} \left[1 - \frac{(x-x_j)}{\Delta}\right]^2 dx = \frac{2\Delta}{3} \quad \text{for } 1 < j < N+1 \\
M_{11} = & \int_{x_1}^{x_2} \left[1 - \frac{(x-x_2)}{\Delta}\right]^2 dx = \frac{\Delta}{3} \\
M_{N+1,N+1} = & \int_{x_N}^{x_{N+1}} \frac{(x-x_N)^2}{\Delta^2} dx = \frac{\Delta}{3}
\end{aligned}$$

Off diagonal components:

$$\begin{aligned}
M_{j,j+1} = & \int_{x_j}^{x_{j+1}} \left[1 - \frac{(x-x_j)}{\Delta}\right] \left[\frac{x-x_j}{\Delta}\right] dx = \frac{\Delta}{6} \quad \text{for } j < N+1 \\
M_{j,j-1} = & \int_{x_{j-1}}^{x_j} \left[\frac{x-x_{j-1}}{\Delta}\right] \left[1 - \frac{(x-x_{j-1})}{\Delta}\right] dx = \frac{\Delta}{6} \quad \text{for } j > 1
\end{aligned}$$

Compute the K_{jk} integrals:

$$\begin{aligned}
K_{jj} = & k'_{jj} \left[\int_{x_{j-1}}^{x_j} \frac{\partial w_j}{\partial x} w_j dx + \int_{x_j}^{x_{j+1}} \frac{\partial w_j}{\partial x} w_j dx\right] \\
= & \frac{1}{2} + -\frac{1}{2} = 0 \quad \text{for } 1 < j < N+1 \\
K_{11} = & -\frac{k'_{11}}{2} = \frac{1}{2}\left[\frac{v}{h} + \frac{w_f}{h\phi}\right] \\
K_{N+1,N+1} = & \frac{k'_{N+1,N+1}}{2} + \min\left[0, \left(\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi_{N+1}}\right)\right] - \frac{D_{in}}{\phi_{N+1}(g_o/h)} \\
= & \left[-\frac{v}{h} - \frac{w_f}{h\phi} + \frac{D_{in}}{\phi^2}\frac{\partial\phi}{\partial x}\right]\frac{1}{2} + \min\left[0, \left(\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi_{N+1}}\right)\right] - \frac{D_{in}}{\phi_{N+1}(g_o/h)}
\end{aligned}$$

Off diagonal components:

$$\begin{aligned}
K_{j(j+1)} = & k'_{j(j+1)} \int_{x_j}^{x_{j+1}} \frac{\partial w_j}{\partial x} w_{j+1} dx = -k'_{j(j+1)} \int_{x_j}^{x_{j+1}} \frac{(x-x_j)}{\Delta^2} dx \\
= & -\frac{k'_{j(j+1)}}{\Delta^2}\frac{\Delta^2}{2} = -\frac{k'_{j(j+1)}}{2} = p5 * \left[\frac{v}{h} + \frac{w_f}{h\phi} - \frac{D_{in}}{\phi^2}\frac{\partial\phi}{\partial x}\right]_{(j+1)} \quad \text{for } j < N+1 \\
K_{j(j-1)} = & k'_{j(j-1)} \int_{x_{j-1}}^{x_j} \frac{\partial w_j}{\partial x} w_{j-1} dx = k'_{j(j-1)} \int_{x_{j-1}}^{x_j} \left[1 - \frac{(x-x_{j-1})}{\Delta^2}\right] dx \\
= & \frac{k'_{j(j-1)}}{\Delta^2}\frac{\Delta^2}{2} = \frac{k'_{j(j-1)}}{2} = -p5 * \left[\frac{v}{h} + \frac{w_f}{h\phi} - \frac{D_{in}}{\phi^2}\frac{\partial\phi}{\partial x}\right]_{(j-1)} \quad \text{for } j > 1
\end{aligned}$$

For $K_{N+1,N}$, there is a boundary contribution:

$$K_{N+1,N} = \frac{k'_{N+1(N)}}{2} - \frac{D_N}{\Delta\phi_N}$$

The bottom condition works if $C_{bot} = h\phi_{N+2}[c]_{ocean}$, ϕ^2 is $\phi_{N+1}\phi_{N+2}$ and

$$\left.\frac{\partial\phi}{\partial x}\right|_{bot} = \frac{\phi_{N+2} - \phi_N}{2\Delta};$$

then the $D_{N+1}/\phi_{N+1}/\Delta$ cancels properly with the porosity gradient. In general

$$\left.\frac{\partial\phi}{\partial x}\right|_k = \frac{\phi_{k+2}-\phi_k}{2\Delta}.$$

When evaluating the integrals for the diffusion term, we will assume that D/ϕ is constant in an element. For $D_{in}/i\phi$ defined on interface points, $D_1 = 0$ and for $j = 2, ..., N$ $D_j/b\phi_j = (D_{in}(j) + D_{in}(j+1))/(i\phi_j + i\phi_{j+1})$. Then the above integrals will be modified as follows:

Compute the S_{jk} integrals:

$$\begin{aligned}
S_{jj} = & -\left[\frac{D_{j-1}}{b\phi_{j-1}}\int_{x_{j-1}}^{x_j}\left(\frac{\partial w_j}{\partial x}\right)^2 dx + \frac{D_j}{b\phi_j}\int_{x_j}^{x_{j+1}}\left(\frac{\partial w_j}{\partial x}\right)^2 dx\right] \\
= & -\frac{1}{\Delta}\left[\frac{D_{j-1}}{b\phi_{j-1}} + \frac{D_j}{b\phi j}\right] \quad \text{for } 1 < j < N+1 \\
S_{11} = & \frac{s'_{11}}{\Delta} = 0 \\
S_{N+1,N+1} = & \frac{s'_{N+1,N+1}}{\Delta} = -\frac{(D_N)}{b\phi_N\Delta}
\end{aligned}$$

Compute the off-diagonal components of S_{jk}:

$$\begin{aligned}
S_{j(j+1)} = & s'_{j(j+1)}\int_{x_j}^{x_{j+1}}\frac{\partial w_j}{\partial x}\frac{\partial w_{j+1}}{\partial x}dx = -\frac{s'_{j(j+1)}}{\Delta} = \frac{D_j}{b\phi_j\Delta} \quad \text{for } j < N+1 \\
S_{j(j-1)} = & s'_{j(j-1)}\int_{x_{j-1}}^{x_j}\frac{\partial w_j}{\partial x}\frac{\partial w_{j-1}}{\partial x}dx = -\frac{s'_{j(j-1)}}{\Delta} = \frac{D_{j-1}}{b\phi_{j-1}} \quad \text{for } j > 1
\end{aligned}$$

We assume that $D/\phi^2\partial\phi/\partial x$ is constant in the element i. If D/ϕ_j is constant, and $\partial\phi/\partial x$ is constant then both the Darcy and D terms go as ϕ^{-1}. Then $\phi = (\phi_j - \phi_{j-1})(x - x_j)/\Delta + \phi_j$ and $m = (\phi_j - \phi_{j-1})/\Delta$ and $b = \phi_j - mx_j$.

The first integral contribution to the Darcy term is:

$$\begin{aligned}
K^1_{jj} = & \frac{-1}{\Delta^2}\left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\int_{j-1}^{j}(x - x_{j-1})\frac{1}{mx+b}dx \\
= & -\left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta^2}\left[\int_{j-1}^{j}\frac{x}{mx+b}dx - x_{j-1}\int_{j-1}^{j}\frac{1}{mx+b}dx\right] \\
= & -\left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta^2}\left[\frac{mx - b\log(b+mx)}{m^2} - x_{j-1}\frac{\log(b+mx)}{m}\right]_{x_{j-1}}^{x_j} \\
= & -\left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta_\phi}\left[1 + \log\left(\frac{\phi_j}{\phi_{j-1}}\right) - \frac{\phi_j}{\Delta_{\phi_j}}\log\left(\frac{\phi_j}{\phi_{j-1}}\right)\right] \\
= & -\left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta_\phi}\left[1 + \frac{\phi_{j-1}}{\Delta_\phi}\log\left(\frac{\phi_j}{\phi_{j-1}}\right)\right]
\end{aligned}$$

$$\begin{aligned}
K^2_{jj} = & \left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta}\int_{x_j}^{x_{j+1}}\left[1 - \frac{(x - x_j)}{\Delta}\right]\frac{1}{mx+b}dx \\
= & \left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta}\left[\frac{(b + m(x_j + \Delta))\log(b+mx) - mx}{\Delta m^2}\right]_{x_j}^{x_{j+1}} \\
= & \left(\frac{w_f}{h} - \frac{D}{\phi}\frac{\partial\phi}{\partial x}\right)\frac{1}{\Delta_\phi}\left[1 - \frac{\phi_{j+1}}{\Delta_\phi}\log\left(\frac{\phi_{j+1}}{\phi_j}\right)\right]
\end{aligned}$$

Now $m = (\phi_{j+1} - \phi_j)/\Delta$ and $b = \phi_{j+1} - mx_{j+1}$.

Source terms $q_{bot} = q_{N+1}$ and $q_{top} = q_1$ (both positive)

$$\begin{aligned} q_{bot} = & \quad \max\left[0, \left(\frac{1}{h}\frac{dh_b}{dt} + \frac{w_f}{h\phi_{N+1}}\right)\right] C|_{bot} + \frac{D_{in}}{\phi_{N+1}(g_o/h)} C|_{bot} \\ C|_{bot} = & \quad \phi_{N+1}[c]_{ocean} \end{aligned}$$

where g_o is not zero.

$$\begin{aligned} q_{in} = & \quad -\min\left[0, \left(\frac{1}{h}\frac{dh_t}{dt} + \frac{w_f}{h\phi}\right) C|_{top}\right] \\ C|_{top} = & \quad h[c]_o\phi_{min} \end{aligned}$$

Calculating the low order solution:

1) Find the lumped mass matrix $M_l = diag\{m_i\}$

$$\begin{aligned} m_j = & \quad \sum_i m_{ji} = m_{j(j+1)} + m_{j(j-1)} + m_{jj} \\ = & \quad \frac{\Delta}{6} + \frac{\Delta}{6} + \frac{2\Delta}{3} = \Delta \quad \text{for } 1 < j < N+1 \\ m_1 = & \quad m_{11} + m_{12} = \frac{\Delta}{3} + \frac{\Delta}{6} = \frac{\Delta}{2} \\ m_{N+1} = & \quad m_{N+1,N} + m_{N+1,N+1} = \frac{\Delta}{6} + \frac{\Delta}{3} = \frac{\Delta}{2} \end{aligned}$$

2) Define artificial diffusion D_a

$$\begin{aligned} d_{j,(j+1)} = & \quad \max\{-k_{j(j+1)}, 0, -k_{(j+1)j}\} = d_{(j+1)j} \\ d_{jj} = & \quad -\sum_{i \neq j} d_{ji} \end{aligned}$$

3) Add artificial diffusion to K: $L = K + D_a$.

4) Solve for the low order predictor solution:

$$(M_l - \Delta t[L + S])C^{n+1} = M_l C^n + \Delta t q$$

Conservations terms for the low order solution are:

$$\begin{aligned} \int \left[C^{n+1} - C^n\right] w(x)dx = & \quad \Delta\left[\frac{c_1^{n+1} - c_1^n}{2} + \frac{c_{N+1}^{n+1} - c_{N+1}^n}{2} + \sum_{j=2}^{N}(c_j^{n+1} - c_j^n)\right] \\ = & \quad \Delta t\left[q_{bot} + q_{in} + (K_{N+1,N+1} + K_{N,N+1})C_{N+1}^{n+1} + (K_{1,1} + K_{2,1})C_1^{n+1}\right] \end{aligned}$$

Now add the antidiffusive flux: compute the F matrix using the low order solution c^{n+1}. Diagonal components are zero. For $i \neq j$

$$f_{ij} = \quad m_{ij}\left[\frac{\Delta c_i}{\Delta t} - \frac{\Delta c_j}{\Delta t} + d_{ij}(c_i^{n+1} - c_j^{n+1}\right].$$

Reaction Equations

The biogeochemical reaction terms for each biogeochemical tracer (see Table *Biogeochemical Tracers* for tracer definitions) are defined in **icepack_algae.F90** in the subroutine *algal_dyn*. The same set of equations is used for the bottom layer model (when `skl_bgc` is true) and the multi-layer biogeochemical model (when `z_tracers` and `solve_zbgc` are true).

Table 4: *Biogeochemical Tracers*

Text Variable	Variable in code	flag	Description	units
N (1)	Nin(1)	*tr_bgc_N*	diatom	$mmol\ N/m^3$
N (2)	Nin(2)	*tr_bgc_N*	small phytoplankton	$mmol\ N/m^3$
N (3)	Nin(3)	*tr_bgc_N*	*Phaeocystis sp*	$mmol\ N/m^3$
DOC (1)	DOCin(1)	*tr_bgc_DOC*	polysaccharids	$mmol\ C/m^3$
DOC (2)	DOCin(2)	*tr_bgc_DOC*	lipids	$mmol\ C/m^3$
DON	DONin(1)	*tr_bgc_DON*	proteins	$mmol\ C/m^3$
fed	Fedin(1)	*tr_bgc_Fe*	dissolved iron	$\mu\ Fe/m^3$
fep	Fepin(1)	*tr_bgc_Fe*	particulate iron	$\mu\ Fe/m^3$
NO_3	Nitin	*tr_bgc_Nit*	NO_3	$mmol\ N/m^3$
NH_4	Amin	*tr_bgc_Am*	NH_4	$mmol\ N/m^3$
SiO_3	Silin	*tr_bgc_Sil*	SiO_2	$mmol\ Si/m^3$
DMSPp	DMSPpin	*tr_bgc_DMS*	particulate DMSP	$mmol\ S/m^3$
DMSPd	DMSPdin	*tr_bgc_DMS*	dissolved DMSP	$mmol\ S/m^3$
DMS	DMSin	*tr_bgc_DMS*	DMS	$mmol\ S/m^3$
PON	PON [a]	*tr_bgc_PON*	passive mobile tracer	$mmol\ N/m^3$
hum	hum [ab]	*tr_bgc_hum*	passive sticky tracer	$mmol\ /m^3$
BC (1)	zaero(1) [a]	*tr_zaero*	black carbon species 1	$kg\ /m^3$
BC (2)	zaero(2) [a]	*tr_zaero*	black carbon species 2	$kg\ /m^3$
dust (1)	zaero(3) [a]	*tr_zaero*	dust species 1	$kg\ /m^3$
dust (2)	zaero(4) [a]	*tr_zaero*	dust species 2	$kg\ /m^3$
dust (3)	zaero(5) [a]	*tr_zaero*	dust species 3	$kg\ /m^3$
dust (4)	zaero(6) [a]	*tr_zaero*	dust species 4	$kg\ /m^3$

[a] not modified in *algal_dyn*

[b] may be in C or N units depending on the ocean concentration

The biochemical reaction term for each algal species has the form:

$$\Delta \mathrm{N}/dt = R_{\mathrm{N}} = \mu(1 - f_{graze} - f_{res}) - M_{ort}$$

where μ is the algal growth rate, M_{ort} is a mortality loss, f_{graze} is the fraction of algal growth that is lost to predatory grazing, and f_{res} is the fraction of algal growth lost to respiration. Algal mortality is temperature dependent and limited by a maximum loss rate fraction (l_{max}):

$$M_{ort} = \min(l_{max}[\mathrm{N}], m_{pre} \exp\{m_T(T - T_{max})\}[\mathrm{N}])$$

Note, $[\cdot]$ denotes brine concentration.

Nitrate and ammonium reaction terms are given by

$$\begin{aligned} \Delta \mathrm{NO}_3/dt = & \quad R_{\mathrm{NO}_3} = [\mathrm{NH}_4]k_{nitr} - U^{tot}_{\mathrm{NO}_3} \\ \Delta \mathrm{NH}_4/dt = & \quad R_{\mathrm{NH}_4} = -[\mathrm{NH}_4]k_{nitr} - U^{tot}_{\mathrm{NH}_4} + (f_{ng} f_{graze}(1 - f_{gs}) + f_{res})\mu^{tot} \\ + & \quad f_{nm} M_{ort} \\ = & \quad -[\mathrm{NH}_4]k_{nitr} - U^{tot}_{\mathrm{NH}_4} + N_{remin} \end{aligned}$$

where the uptake U^{tot} and algal growth μ^{tot} are accumulated totals for all algal species. k_{nitr} is the nitrification rate and f_{ng} and f_{nm} are the fractions of grazing and algal mortality that are remineralized to ammonium and f_{gs} is the fraction of grazing spilled or lost. Algal growth and nutrient uptake terms are described in more detail in *Algal growth and nutrient uptake*.

Dissolved organic nitrogen satisfies the equation

$$\Delta\mathrm{DON}/dt = \quad R_{\mathrm{DON}} = f_{dg} f_{gs} f_{graze} \mu^{tot} - [\mathrm{DON}] k_{nb}$$

With a loss from bacterial degration (rate k_{nb}) and a gain from spilled grazing that does not enter the NH_4 pool.

A term Z_{oo} closes the nitrogen cycle by summing all the excess nitrogen removed as zooplankton/bacteria in a timestep. This term is not a true tracer, i.e. not advected horizontally with the ice motion, but provides a diagnostic comparison of the amount of N removed biogeochemically from the ice N-NO_3-NH_4-DON cycle at each timestep.

$$\mathrm{Z}_{oo} = \quad [(1 - f_{ng}(1 - f_{gs}) - f_{dg} f_{gs}] f_{graze} \mu^{tot} dt + (1 - f_{nm}) M_{ort} dt + [\mathrm{DON}] k_{nb} dt$$

Dissolved organic carbon may be divided into polysaccharids and lipids. Parameters are two dimensional (indicated by superscript i) with index 1 corresponding to polysaccharids and index 2 appropriate for lipids. The DOC^i equation is:

$$\Delta\mathrm{DOC}^i/dt = \quad R_{\mathrm{DOC}} = f^i_{cg} f_{ng} \mu^{tot} + R^i_{c:n} M_{ort} - [\mathrm{DOC}] k^i_{cb}$$

Silicate has no biochemical source terms within the ice and is lost only through algal uptake:

$$\Delta\mathrm{SiO}_3/dt = \quad R_{\mathrm{SiO}_3} = -U^{tot}_{\mathrm{SiO}_3}$$

Dissolved iron has algal uptake and remineralization pathways. In addition, fed may be converted to or released from the particulate iron pool depending on the dissolve iron (fed) to polysaccharid (DOC(1)) concentration ratio. If this ratio exceeds a maximum value $r^{max}_{fed:doc}$ then the change in concentration for dissolved and particulate iron is

$$\begin{aligned} \Delta_{fe}\mathrm{fed}/dt = &\quad -[\mathrm{fed}]/\tau_{fe} \\ \Delta_{fe}\mathrm{fep}/dt = &\quad [\mathrm{fed}]/\tau_{fe} \end{aligned}$$

For values less than $r^{max}_{fed:doc}$

$$\begin{aligned} \Delta_{fe}\mathrm{fed}/dt = &\quad [\mathrm{fep}]/\tau_{fe} \\ \Delta_{fe}\mathrm{fep}/dt = &\quad -[\mathrm{fep}]/\tau_{fe} \end{aligned}$$

Very long timescales τ_{fe} will remove this source/sink term. The default value is currently set at 3065 days to turn off this dependency (any large number will do to turn it off). 61-65 days is a more realistic option (Parekh et al., 2004).

The full equation for fed including uptake and remineralization is

$$\Delta\mathrm{fed}/dt = \quad R_{\mathrm{fed}} = -U^{tot}_{\mathrm{fed}} + f_{fa} R_{fe:n} N_{remin} + \Delta_{fe}\mathrm{fed}/dt$$

Particulate iron also includes a source term from algal mortality and grazing that is not immediately bioavailable. The full equation for fep is

$$\Delta\mathrm{fep}/dt = \quad R_{\mathrm{fep}} = R_{fe:n}[\mathrm{Z}_{oo}/dt + (1 - f_{fa})] N_{remin} + \Delta_{fe}\mathrm{fep}/dt$$

The sulfur cycle includes DMS and dissolved DMSP (DMSPd). Particulate DMSP is assumed to be proportional to the algal concentration, i.e. DMSPp $= R^i_{s:n}\mathrm{N}^i$ for algal species i. For DMSP and DMS,

$$\begin{aligned} \Delta\mathrm{DMSPd}/dt = &\quad R_{\mathrm{DMSPd}} = R_{s:n}[f_{sr} f_{res} \mu^{tot} + f_{nm} M_{ort}] - [\mathrm{DMSPd}]/\tau_{dmsp} \\ \Delta\mathrm{DMS}/dt = &\quad R_{\mathrm{DMS}} = y_{dms}[\mathrm{DMSPd}]/\tau_{dmsp} - [\mathrm{DMS}]/\tau_{dms} \end{aligned}$$

See *BGC Tuning Parameters* for a more complete list and description of biogeochemical parameters.

Algal growth and nutrient uptake

Nutrient limitation terms are defined in the simplest ecosystem for NO_3. If the appropriate tracer flags are true, then limitation terms may also be found for NH_4, SiO_3, and fed

$$\begin{aligned}
\mathrm{NO}_{3lim} &= \frac{[\mathrm{NO}_3]}{[\mathrm{NO}_3] + K_{\mathrm{NO}_3}} \\
\mathrm{NH}_{4lim} &= \frac{[\mathrm{NH}_4]}{[\mathrm{NH}_4] + K_{\mathrm{NH}_4}} \\
N_{lim} &= \min(1, \mathrm{NO}_{3lim} + \mathrm{NH}_{4lim}) \\
\mathrm{SiO}_{3lim} &= \frac{[\mathrm{SiO}_3]}{[\mathrm{SiO}_3] + K_{\mathrm{SiO}_3}} \\
\mathrm{fed}_{lim} &= \frac{[\mathrm{fed}]}{[\mathrm{fed}] + K_{\mathrm{fed}}}
\end{aligned}$$

Light limitation L_{lim} is defined in the following way: $I_{sw}(z)$ (in W/m^2) is the shortwave radiation at the ice level and the optical depth is proportional to the chlorophyll concentration, $op_{dep} =$ `chlabs` [Chl*a*]. If ($op_{dep} > op_{min}$) then

$$I_{avg} = I_{sw}(1 - \exp(-op_{dep}))/op_{dep}$$

otherwise $I_{avg} = I_{sw}$.

$$L_{lim} = (1 - \exp(-\alpha I_{avg}))\exp(-\beta I_{avg})$$

The maximal algal growth rate before limitation is

$$\begin{aligned}
\mu_o &= \mu_{max}\exp(\mu_T \Delta T) f_{sal}[\mathrm{N}] \\
\mu' &= min(L_{lim}, N_{lim}, \mathrm{SiO}_{3lim}, \mathrm{fed}_{lim})\mu_o
\end{aligned}$$

where μ' is the initial estimate of algal growth rate for a given algal species and ΔT is the difference between the local tempurature and the maximum (in this case $\mathrm{T}_{max} = 0^o\mathrm{C}$).

The initial estimate of the uptake rate for silicate and iron is

$$\begin{aligned}
\tilde{U}_{\mathrm{SiO}_3} &= R_{si:n}\mu' \\
\tilde{U}_{\mathrm{fed}} &= R_{fe:n}\mu'
\end{aligned}$$

For nitrogen uptake, we assume that ammonium is preferentially acquired by algae. To determine the nitrogen uptake needed for each algal growth rate of μ, first determine the "potential" uptake rate of ammonium:

$$U'_{\mathrm{NH}_4} = \mathrm{NH}_{4lim}\mu_o$$

Then

$$\begin{aligned}
\tilde{U}_{\mathrm{NH}_4} &= \min(\mu', U'_{\mathrm{NH}_4}) \\
\tilde{U}_{\mathrm{NO}_3} &= \mu' - \tilde{U}_{\mathrm{NH}_4}
\end{aligned}$$

We require that each rate not exceed a maximum loss rate l_{max}/dt. This is particularly important when multiple species are present. In this case, the accumulated uptake rate for each nutrient is found and the fraction (fU^i) of uptake due to algal species i is saved. Then the total uptake rate is compared with the maximum loss condition. For example, the net uptake of nitrate when there are three algal species is

$$\tilde{U}^{tot}_{\mathrm{NO}_3} = \sum_{i=1}^{3} \tilde{U}^i_{\mathrm{NO}_3} \quad .$$

Then the uptake fraction for species i and the adjusted total uptake is

$$\begin{aligned} fU^i_{\mathrm{NO_3}} &= \frac{\tilde{U}^i_{\mathrm{NO_3}}}{\tilde{U}^{tot}_{\mathrm{NO_3}}} \\ U^{tot}_{\mathrm{NO_3}} &= \min(\tilde{U}^{tot}_{\mathrm{NO_3}}, l_{max}[\mathrm{NO_3}]/dt) \end{aligned}$$

Now, for each algal species the nitrate uptake is

$$U^i_{\mathrm{NO_3}} = fU^i_{\mathrm{NO_3}} U^{tot}_{\mathrm{NO_3}}$$

Similar expressions are found for all potentially limiting nutrients. Then the true growth rate for each algal species i is

$$\mu^i = \min(U^i_{\mathrm{SiO_3}}/R_{si:n}, U^i_{\mathrm{NO_3}} + U^i_{\mathrm{NH_4}}, U^i_{\mathrm{fed}}/R_{fe:n})$$

Preferential ammonium uptake is assumed once again and the remaining nitrogen is taken from the nitrate pool.

CHAPTER

THREE

STANDALONE USER GUIDE

3.1 Implementation

Icepack is written in FORTRAN90 and runs on platforms using UNIX, LINUX, and other operating systems. The code is not parallelized. (CHANGE IF OPENMP IS IMPLEMENTED)

Icepack consists of the sea ice column physics code, contained in the **columnphysics/** directory, and a **configuration/** directory that includes a driver for testing the column physics and a set of scripts for configuring the tests. Icepack is designed such that the column physics code may be used by a host sea ice model without direct reference to the driver or scripts, although these may be consulted for guidance when coupling the column physics code to the host sea ice model (CICE may also be useful for this.) Information about the interface between the column physics and the driver or host sea ice model is located in the *Initialization and Forcing* section.

3.1.1 Directory structure

The present code distribution includes source code for the column physics, source code for the driver, and the scripts. Forcing data is available from the ftp site. The directory structure of Icepack is as follows. All columnphysics filename have a prefix of icepack_ and all driver files are prefixed with icedrv_*.

LICENSE.pdf
license for using and sharing the code

DistributionPolicy.pdf
policy for using and sharing the code

README.md
basic information and pointers

columnphysics/
columnphysics source code, see *Icepack Column Physics*

configuration/scripts/
support scripts, see *Scripts Implementation*

configuration/driver/
icepack driver code, see *Driver Implementation*

doc/
documentation

icepack.setup
main icepack script for creating cases

A case (compile) directory is created upon initial execution of the script **icepack.setup** at the user-specified location provided after the -c flag. Executing the command `./icepack.setup -h` provides helpful information for this tool.

3.1.2 Grid and boundary conditions

The driver configures a collection of grid cells on which the column physics code will be run. This "horizontal" grid is a vector of length `nx`, with a minimum length of 4. The grid vector is initialized with different sea ice conditions, such as open water, a uniform slab of ice, a multi-year ice thickness distribution with snow, and land. For simplicity, the same forcing values are applied to all grid cells.

Icepack includes two vertical grids. The basic vertical grid contains `nilyr` equally spaced grid cells. History variables available for column output are ice and snow temperature, `Tinz` and `Tsnz`. These variables also include thickness category as a fourth dimension.

In addition, there is a bio-grid that can be more finely resolved and includes additional nodes for boundary conditions. It is used for solving the brine height variable h_b and for discretizing the vertical transport equations of biogeochemical tracers. The bio-grid is a non-dimensional vertical grid which takes the value zero at h_b and one at the ice–ocean interface. The number of grid levels is specified during compilation by setting the variable `NBGCLYR` equal to an integer (n_b) .

Ice tracers and microstructural properties defined on the bio-grid are referenced in two ways: as `bgrid` $= n_b + 2$ points and as igrid$= n_b + 1$ points. For both bgrid and igrid, the first and last points reference h_b and the ice–ocean interface, respectively, and so take the values 0 and 1, respectively. For bgrid, the interior points $[2, n_b + 1]$ are spaced at $1/n_b$ intervals beginning with *bgrid(2)* = $1/(2n_b)$. The `igrid` interior points $[2, n_b]$ are also equidistant with the same spacing, but physically coincide with points midway between those of `bgrid`.

3.1.3 Initialization and Forcing

Icepack's parameters and variables are initialized in several steps. Many constants and physical parameters are set in **icepack_parameters.F90**. In the current driver implementation, a namelist file is read to setup the model. Namelist values are given default values in the code, which may then be changed when the input file **icepack_in** is read. Other physical constants, numerical parameters, and variables are first set in initialization routines for each ice model component or module. Then, if the ice model is being restarted from a previous run, core variables are read and reinitialized in *restartfile*, while tracer variables needed for specific configurations are read in separate restart routines associated with each tracer or specialized parameterization. Finally, albedo and other quantities dependent on the initial ice state are set. Some of these parameters will be described in more detail in the *Tables of Namelist Options*.

Two namelist variables control model initialization, `ice_ic` and `restart`. Setting `ice_ic` = 'default' causes the model to run using initial values set in the code. To start from a file **filename**, set `restart` = .true. and `ice_ic` = **filename**. When restarting using the Icepack driver, for simplicity the tracers are assumed to be set the same way (on/off) as in the run that created the restart file; i.e. that the restart file contains exactly the information needed for the new run. CICE is more flexible in this regard.

For stand-alone runs, routines in **icedrv_forcing.F90** read and interpolate data from files, and are intended merely for testing, although they can also provide guidance for the user to write his or her own routines.

3.1.4 Choosing an appropriate time step

Transport in thickness space imposes a restraint on the time step, given by the ice growth/melt rate and the smallest range of thickness among the categories, $\Delta t < \min(\Delta H)/2\max(f)$, where ΔH is the distance between category boundaries and f is the thermodynamic growth rate. For the 5-category ice thickness distribution used as the default in this distribution, this is not a stringent limitation: $\Delta t < 19.4$ hr, assuming $\max(f) = 40$ cm/day.

3.1.5 Model output

The Icepack model provides diagnostic output files, binary or netCDF restart files, and a primitive netCDF history file capability. The sea ice model CICE provides more extensive options for model output, including many derived output variables.

Diagnostic files

Icepack writes diagnostic information for each grid cell as a separate file, **ice_diag.***, identified by the initial ice state of the grid cell (no ice, slab, land, etc).

Restart files

Icepack provides restart data in binary unformatted format or netCDF. The restart files created by the Icepack driver contain all of the variables needed for a full, exact restart. The filename begins with the character string 'iced.' and is placed in the directory specified by the namelist variable `restart_dir`. The restart dump frequency is given by the namelist variable `dumpfreq`. The namelist variable `ice_ic` contains the pointer to the filename from which the restart data is to be read and the namelist option `restart` must be set to `.true.` to use the file. `dump_last` namelist can also be set to true to trigger restarts automatically at the end of runs. The default restart file format is binary, set in namelist with `restart_format` = 'bin'. For netCDF, set `restart_format` = 'nc' or use `icepack.setup -s restcdf`.

The default configuration of Icepack does not support netCDF. If netCDF restart files are desired, the USE_NETCDF C preprocessor directive must be set during compilation. This is done by setting `ICE_IOTYPE` to `netcdf` in **icepack.settings** or using the `icepack.setup -s` option `ionetcdf`. If netCDF is used on a particular machine, the machine env and Macros file must support compilation with netCDF.

History files

Icepack has a primitive netCDF history capability that is turned on with the `history_format` namelist. When `history_format` is set to 'nc', history files are created for each run with a naming convention of **icepack.h.yyyymmdd.nc** in the run directory history directory. The yyyymmdd is the start date for each run. Use `icepack.setup -s histcdf` to turn on netCDF history files automatically.

When Icepack history files are turned on, data for a set of fixed fields is written to the history file for each column at every timestep without ability to control fields, frequencies, or temporal averaging. All output fields are hardwired into the implementation in **configuration/driver/icedrv_history.F90** file. The netCDF file does NOT meet netCDF CF conventions and is provided as an amenity in the standalone Icepack model. Users are free to modify the output fields or extend the implementation and are encouraged to share any updates with the Consortium.

The default configuration of Icepack does not support netCDF. If netCDF history files are desired, the USE_NETCDF C preprocessor directive must be set during compilation. This is done by setting `ICE_IOTYPE` to `netcdf` in **icepack.settings** or using the `icepack.setup -s` option `ionetcdf`. If netCDF is used on a particular machine, the machine env and Macros file must support compilation with netCDF.

Biogeochemistry History Fields

History output is not provided with Icepack. This documentation indicates what is available for output and is implemented in CICE.

Table *Biogeochemical History variables* lists the biogeochemical tracer history flags along with a short description and the variable or variables saved. Not listed are flags appended with _ai, i.e. f_fbio_ai. These fields are identical to their counterpart. i.e. f_fbio, except they are averaged by ice area.

Table 1: *Biogeochemical History variables*

History Flag	Definition	Variable(s)	Units
f_fiso_atm	atmospheric water isotope deposition flux	fiso_atm	kg m^{-2} s^{-1}
f_fiso_ocn	water isotope flux from ice to ocean	fiso_ocn	kg m^{-2} s^{-1}
f_iso	isotope mass (snow and ice)	isosno, isoice	kg/kg
f_faero_atm	atmospheric aerosol deposition flux	faero_atm	kg m^{-2} s^{-1}
f_faero_ocn	aerosol flux from ice to ocean	faero_ocn	kg m^{-2} s^{-1}
f_aero	aerosol mass (snow and ice ssl and int)	aerosnossl, aerosnoint,aeroicessl, aeroiceint	kg/kg
f_fbio	biological ice to ocean flux	fN, fDOC, fNit, fAm,fDON,fFep^a, fFed^a, fSil,fhum, fPON, fDMSPd,fDMS, fDMSPp, fzaero	mmol m^{-2} s^{-1}
f_zaero	bulk z-aerosol mass fraction	zaero	kg/kg
f_bgc_S	DEPRECATED	bgc_S	ppt
f_bgc_N	bulk algal N concentration	bgc_N	mmol m^{-3}
f_bgc_C	bulk algal C concentration	bgc_C	mmol m^{-3}
f_bgc_DOC	bulk DOC concentration	bgc_DOC	mmol m^{-3}
f_bgc_DON	bulk DON concentration	bgc_DON	mmol m^{-3}
f_bgc_DIC	bulk DIC concentration	bgc_DIC	mmol m^{-3}
f_bgc_chl	bulk algal chlorophyll concentration	bgc_chl	mg chl m^{-3}
f_bgc_Nit	bulk nitrate concentration	bgc_Nit	mmol m^{-3}
f_bgc_Am	bulk ammonium concentration	bgc_Am	mmol m^{-3}
f_bgc_Sil	bulk silicate concentration	bgc_Sil	mmol m^{-3}
f_bgc_DMSPp	bulk particulate DMSP concentration	bgc_DMSPp	mmol m^{-3}
f_bgc_DMSPd	bulk dissolved DMSP concentration	bgc_DMSPd	mmol m^{-3}
f_bgc_DMS	bulk DMS concentration	bgc_DMS	mmol m^{-3}
f_bgc_Fe	bulk dissolved and particulate iron conc.	bgc_Fed, bgc_Fep	μ mol m^{-3}
f_bgc_hum	bulk humic matter concentration	bgc_hum	mmol m^{-3}
f_bgc_PON	bulk passive mobile tracer conc.	bgc_PON	mmol m^{-3}
f_upNO	Total algal NO_3 uptake rate	upNO	mmol m^{-2} d^{-1}
f_upNH	Total algal NH_4 uptake rate	upNH	mmol m^{-2} d^{-1}
f_bgc_ml	upper ocean tracer concentrations	ml_N, ml_DOC, ml_Nit,ml_Am, ml_DON, ml_Fep^b,ml_Fed^b, ml_Sil, ml_hum, ml_PON,ml_DMS, ml_DMSPd, ml_DMSPp	mmol m^{-3}
f_bTin	ice temperature on the bio grid	bTizn	^{o}C
f_bphi	ice porosity on the bio grid	bphizn	%
f_iDin	brine eddy diffusivity on the interface bio grid	iDin	m^2 d^{-1}

continues on next page

Table 1 – continued from previous page

History Flag	Definition	Variable(s)	Units
f_iki	ice permeability on the interface bio grid	ikin	mm^2
f_fbri	ratio of brine tracer height to ice thickness	fbrine	1
f_hbri	brine tracer height	hbrine	m
f_zfswin	internal ice PAR on the interface bio grid	zfswin	W m^{-2}
f_bionet	brine height integrated tracer concentration	algalN_net, algalC_net, chl_net, pFe^c_net, dFe^c_net, Sil_net, Nit_net, Am_net, hum_net, PON_net, DMS_net, DMSPd_net, DMSPp_net, DOC_net, zaero_net, DON_net	mmol m^{-2}
f_biosnow	snow integrated tracer concentration”	algalN_snow, algalC_snow,chl_snow, pFe^c_snow, dFe^c_snow,Sil_snow, Nit_snow, Am_snow, hum_snow, PON_snow, DMS_snow, DMSPd_snow, DMSPp_snow, DOC_snow, zaero_snow, DON_snow	mmol m^{-2}
f_grownet	Net specific algal growth rate	grow_net	m d^{-1}
f_PPnet	Net primary production	PP_net	mgC m^{-2} d^{-1}
f_algalpeak	interface bio grid level of peak chla	peak_loc	1
f_zbgc_frac	mobile fraction of tracer	algalN_frac, chl_frac, pFe_frac,dFe_frac, Sil_frac, Nit_frac,Am_frac, hum_frac, PON_frac,DMS_frac, DMSPd_frac, DMSPp_frac,DOC_frac, zaero_frac, DON_frac	1

[a] units are μmol m^{-2} s^{-1}

[b] units are μmol m^{-3}

[c] units are μmol m^{-2}

3.2 Running Icepack

Quick-start instructions are provided in the *Quick Start* section.

3.2.1 Scripts

The icepack scripts are written to allow quick setup of cases and tests. Once a case is generated, users can manually modify the namelist and other files to custom configure the case. Several settings are available via scripts as well.

Overview

Most of the scripts that configure, build and run Icepack are contained in the directory **configuration/scripts/**, except for **icepack.setup**, which is in the main directory. **icepack.setup** is the main script that generates a case.

Users may need to port the scripts to their local machine. Specific instructions for porting are provided in *Porting*.

`icepack.setup -h` will provide the latest information about how to use the tool. `icepack.setup --help` will provide an extended version of the help. There are three usage modes,

- `--case` or `-c` creates individual stand alone cases.
- `--test` creates individual tests. Tests are just cases that have some extra automation in order to carry out particular tests such as exact restart.
- `--suite` creates a test suite. Test suites are predefined sets of tests and `--suite` provides the ability to quickly setup, build, and run a full suite of tests.

All modes will require use of `--mach` or `-m` to specify the machine. Use of `--env` is also recommended to specify the compilation environment. `--case` and `--test` modes can use `--set` or `-s` which will turn on various model options. `--test` and `--suite` will require `--testid` to be set and can use `--bdir`, `--bgen`, `--bcmp`, and `--diff` to generate (save) results for regression testing (comparison with prior results). `--tdir` will specify the location of the test directory. Testing will be described in greater detail in the *Testing Icepack* section.

Again, `icepack.setup --help` will show the latest usage information including the available `--set` options, the current ported machines, and the test choices.

To create a case, run **icepack.setup**:

```
icepack.setup -c mycase -m machine -e intel
cd mycase
```

Once a case/test is created, several files are placed in the case directory

- **env.[machine]_[env]** defines the machine environment
- **icepack.settings** defines many variables associated with building and running the model
- **makdep.c** is a tool that will automatically generate the make dependencies
- **Macros.[machine]_[env]** defines the Makefile macros
- **Makefile** is the makefile used to build the model
- **icepack.build** is a script that builds and compiles the model
- **icepack_in** is the namelist input file
- **icepack.run** is a batch run script
- **icepack.submit** is a simple script that submits the icepack.run script

All scripts and namelist are fully resolved in the case. Users can edit any of the files in the case directory manually to change the model configuration, build options, or batch settings. The file dependency is indicated in the above list. For instance, if any of the files before **icepack.build** in the list are edited, **icepack.build** should be rerun.

The **casescripts/** directory holds scripts used to create the case and can largely be ignored. Once a case is created, the **icepack.build** script should be run interactively and then the case should be submitted by executing the **icepack.submit** script interactively. The **icepack.submit** script submits the **icepack.run** or **icepack.test** script.

Some hints:

- To change namelist, manually edit the **icepack_in** file
- To change batch settings, manually edit the top of the **icepack.run** or **icepack.test** (if running a test) file

- When the run scripts are submitted, the current **icepack_in**, **icepack.settings**, and **env.[machine]** files are copied from the case directory into the run directory. Users should generally not edit files in the run directory as these are overwritten when following the standard workflow. **icepack.settings** can be sourced to establish the case values in the login shell. An alias like the following can be established to quickly switch between case and run directories:

```
alias  cdrun 'cd `\grep "setenv ICE_RUNDIR"  icepack.settings | awk "{print "\$"NF}
↪"`'
alias cdcase 'cd `\grep "setenv ICE_CASEDIR" icepack.settings | awk "{print "\$"NF}
↪"`'
```

- To turn on the debug compiler flags, set `ICE_BLDDEBUG` in **icepack.setttings** to true
- To change compiler options, manually edit the Macros file. To add user defined preprocessor macros, modify `ICE_CPPDEFS` in **icepack.settings** using the syntax `-DCICE_MACRO`.
- To clean the build before each compile, set `ICE_CLEANBUILD` in **icepack.settings** to true. To not clean before the build, set `ICE_CLEANBUILD` in **icepack.settings** to false

To build and run:

```
./icepack.build
./icepack.submit
```

The build and run log files will be copied into the logs subdirectory in the case directory. Other model output will be in the run directory. The run directory is set in **icepack.settings** via the **ICE_RUNDIR** variable. To modify the case setup, changes should be made in the case directory, NOT the run directory.

Command Line Options

`icepack.setup -h` provides a summary of the command line options. There are three different modes, `--case`, `--test`, and `--suite`. This section provides details about the relevant options for setting up cases with examples. Testing will be described in greater detail in the *Testing Icepack* section.

`--help`, `-h`
: prints `icepack.setup` help information to the terminal and exits.

`--version`
: prints the Icepack version to the terminal and exits.

`--setvers`
: Updates the stored value of the Icepack version in the sandbox and exits See *Model Version Control* for more information.

`--docintfc`
: Runs a script that updates the public interfaces in the documentation. This script parses the source code directly. See *Public Interfaces* for more information.

`--case`, `-c` CASE
: specifies the case name. This can be either a relative path of an absolute path. This cannot be used with –test or –suite. Either `--case`, `--test`, or `--suite` is required.

`--mach`, `-m` MACHINE
: specifies the machine name. This should be consistent with the name defined in the Macros and env files in **configurations/scripts/machines**. This is required in all modes and is paired with `--env` to define the compilation environment.

`--env`, `-e` ENVIRONMENT1,ENVIRONMENT2,ENVIRONMENT3
: specifies the compilation environment associated with the machine. This should be consistent with the name defined in the Macros and env files in **configurations/scripts/machines**. Each machine can have multiple supported environments including support for different compilers, different compiler versions, different mpi libraries, or other system settigs. When used with `--suite` or `--test`, the ENVIRONMENT can be a set of comma deliminated values with no spaces and the tests will then be run for all of those environments. With `--case`, only one ENVIRONMENT should be specified. (default is intel)

`--pes`, `-p` MxN
: specifies the number of tasks and threads the case should be run on. This only works with `--case`. The format is tasks x threads or "M"x"N" where M is tasks and N is threads and both are integers. The current icepack driver is purely serial so setting multiple tasks or multiple threads will have no impact. (default is 1x1)

`--acct` ACCOUNT
: specifies a batch account number. This is optional. See *Machine Account Settings* for more information.

`--queue` QUEUE
: specifies a batch queue name. This is optional. See *Machine Queue Settings* for more information.

`--grid`, `-g` GRID
: specifies the grid. This is a string and for the current icepack driver, only col is supported. (default = col)

`--set`, `-s` SET1,SET2,SET3
: specifies the optional settings for the case. This is only used with `--case` or `--test`. The settings for `--suite` are defined in the suite file. Multiple settings can be specified by providing a comma deliminated set of values without spaces between settings. The available settings are in **configurations/scripts/options** and `icepack.setup --help` will also list them. These settings files can change either the namelist values or overall case settings (such as the debug flag).

For Icepack, when setting up cases, the `--case` and `--mach` must be specified. It's also recommended that `--env` be set explicitly as well. At the present time, `--pes` and `--grid` cannot vary from 1x1 and col respectively which are the defaults. `--acct` is not normally used. A more convenient method is to use the **~/cice_proj** file, see *Machine Account Settings*. The `--set` option can be extremely handy. The `--set` options are documented in *Preset Options*.

Preset Options

There are several preset options. These are hardwired in **configurations/scripts/options** and are specfied for a case or test by the `--set` command line option. You can see the full list of settings by doing `icepack.setup --help`.

The default icepack namelist and icepack settings are specified in the files **configuration/scripts/icepack_in** and **configuration/scripts/icepack.settings** respectively. When picking a preset setting (option), the set_env.setting and set_nml.setting will be used to change the defaults. This is done as part of the `icepack.setup` and the modifications are resolved in the **icepack.settings** and **icepack_in** file placed in the case directory. If multiple options are chosen that conflict, then the last option chosen takes precedence. Not all options are compatible with each other.

Some of the options are

`debug` which turns on the compiler debug flags

`short`, `medium`, `long` which change the batch time limit

`diag1` which turns on diagnostics each timestep

`leap` which turns on the leap year

`pondlvl`, `pondtopo` which turn on the various pond schemes

`run10day`, `run1year`, etc which specifies a run length

`swccsm3` which turns on the ccsm3 shortwave and albedo computation `thermo1` which on turns on the Bitz-Lipscomb thermodynamics model (default is mushy-layer)

`bgc*` which turns of various bgc configurations

and there are others. To add a new option, just add the appropriate file in **configuration/scripts/options**. Some of the options settings like `smoke` and `restart` are specifically geared toward setting up tests. For more information, see *Preset Case Options*

Examples

The simplest case is just to setup a default configurations specifying the case name, machine, and environment:

```
icepack.setup --case mycase1 --mach spirit --env intel
```

To add some optional settings, one might do:

```
icepack.setup --case mycase2 --mach spirit --env intel --set debug,diag1,run1year,
↪pondtopo
```

Once the cases are created, users are free to modify the **icepack.settings** and **icepack_in** namelist to further modify their setup.

C Preprocessor (CPP) Macros

There are a few C Preprocessor Macros supported in the Icepack model. These support certain coding features to be excluded or included during the compile. They exist in part to support the CICE model and other applications that use Icepack.

For standalone Icepack, The CPPs are defined by the *CPPDEFS* variable in the Icepack Makefile. They are defined by passing the -D[CPP] to the C and Fortran compilers (ie. -DNO_I8) and this is what needs to be set in the `CPPDEFS` variable. The value of `ICE_CPPDEFS` in **icepack.settings** is copied into the Makefile `CPPDEFS` variable as are settings hardwired into the **Macros.[machine]_[environment]** file.

A list of available CPPs can be found in *Table of C Preprocessor (CPP) Macros*.

Model Version Control

Managing the internal representation of the model version is handled through the **icepack.setup** script. The `--version` option displays the version value on the terminal. The `--setvers` option updates the version defined in the sandbox. It is highly recommended that any changes to the version name be done through this interface to make sure it's done correctly and comprehensively. The version name should just include the string associated with the major, minor, and similar. For instance,:

```
icepack.setup --version
```

returns

> ./icepack.setup: This is ICEPACK_v1.0.0.d0003

and:

```
icepack.setup --setvers v1.0.0.d0004
```

would update the version. Always check the string by doing `icepack.setup --version` after invoking `icepack.setup --setvers`.

The version is not updated in the repository unless the code changes associated with the new version are pushed to the repository.

Other Scripts Tools

There are other scripts that come with icepack. These include

- setup_run_dirs.csh. This scripts is added to the case directory. Invoking it creates all the run directories manually. This script is automatically called as part of the run script, but sometimes it's useful to create these directories before submitting in order to stage custom input files or other data.

3.2.2 Porting

To port, an **env.[machine]_[environment]** and **Macros.[machine]_[environment]** file have to be added to the **configuration/scripts/machines/** directory and the **configuration/scripts/icepack.batch.csh** file needs to be modified. In addition **configuration/scripts/icepack.launch.csh** may need to be modified if simply running the binary directly will not work. In general, the machine is specified in `icepack.setup` with `--mach` and the environment (compiler) is specified with `--env`. mach and env in combination define the compiler, compiler version, supporting libaries, and batch information. Multiple compilation environments can be created for a single machine by choosing unique env names.

- cd to **configuration/scripts/machines/**
- Copy an existing env and a Macros file to new names for your new machine
- Edit your env and Macros files, update as needed
- cd .. to **configuration/scripts/**
- Edit the **icepack.batch.csh** script to add a section for your machine with batch settings and job launch settings
- Edit the **icepack.launch.csh** script to add a section for your machine if executing the binary directly is not supported
- Download and untar a forcing dataset to the location defined by `ICE_MACHINE_INPUTDATA` in the env file

In fact, this process almost certainly will require some iteration. The easiest way to carry this out is to create an initial set of changes as described above, then create a case and manually modify the **env.[machine]** file and **Macros.[machine]** file until the case can build and run. Then copy the files from the case directory back to **configuration/scripts/machines/** and update the **configuration/scripts/icepack.batch.csh** file, retest, and then add and commit the updated machine files to the repository.

Machine variables

There are several machine specific variables defined in the **env.$[machine]**. These variables are used to generate working cases for a given machine, compiler, and batch system. Some variables are optional.

Table 2: *Machine Settings*

variable	format	description
ICE_MACHINE_MACHNAME	string	machine name
ICE_MACHINE_MACHINFO	string	machine information
ICE_MACHINE_ENVNAME	string	env/compiler name
ICE_MACHINE_ENVINFO	string	env/compiler information
ICE_MACHINE_MAKE	string	make command
ICE_MACHINE_WKDIR	string	root work directory
ICE_MACHINE_INPUTDATA	string	root input data directory
ICE_MACHINE_BASELINE	string	root regression baseline directory
ICE_MACHINE_SUBMIT	string	batch job submission command
ICE_MACHINE_TPNODE	integer	machine maximum MPI tasks per node
ICE_MACHINE_ACCT	string	batch default account
ICE_MACHINE_QUEUE	string	batch default queue
ICE_MACHINE_BLDTHRDS	integer	number of threads used during build
ICE_MACHINE_QSTAT	string	batch job status command (optional)
ICE_MACHINE_QUIETMODE	true/false	flag to reduce build output (optional)

Cross-compiling

It can happen that the model must be built on a platform and run on another, for example when the run environment is only available in a batch queue. The program **makdep** (see *Overview*), however, is both compiled and run as part of the build process.

In order to support this, the Makefile uses a variable `CFLAGS_HOST` that can hold compiler flags specfic to the build machine for the compilation of makdep. If this feature is needed, add the variable `CFLAGS_HOST` to the **Macros.[machine]_[environment]** file. For example :

```
CFLAGS_HOST = -xHost
```

Machine Account Settings

The machine account default is specified by the variable `ICE_MACHINE_ACCT` in the **env.[machine]** file. The easiest way to change a user's default is to create a file in your home directory called **.cice_proj** and add your preferred account name to the first line. There is also an option (`--acct`) in **icepack.setup** to define the account number. The order of precedence is **icepack.setup** command line option, **.cice_proj** setting, and then value in the **env.[machine]** file.

Machine Queue Settings

The machine queue default is specified by the variable `ICE_MACHINE_QUEUE` in the **env.[machine]** file. The easiest way to change a user's default is to create a file in your home directory called **.cice_queue** and add your preferred account name to the first line. There is also an option (`--queue`) in **icepack.setup** to define the queue name on a case basis. The order of precedence is **icepack.setup** command line option, **.cice_queue** setting, and then value in the **env.[machine]** file.

3.2.3 Porting to Laptop or Personal Computers

To get the required software necessary to build and run Icepack, a conda environment file is available at :

`configuration/scripts/machines/environment.yml`.

This configuration is supported by the Consortium on a best-effort basis on macOS and GNU/Linux. It is untested under Windows, but might work using the Windows Subsystem for Linux.

Once you have installed Miniconda and created the `icepack` conda environment by following the procedures in this section, Icepack should run on your machine without having to go through the formal *Porting* process outlined above.

Installing Miniconda

We recommend the use of the Miniconda distribution to create a self-contained conda environment from the `environment.yml` file. This process has to be done only once. If you do not have Miniconda or Anaconda installed, you can install Miniconda by following the official instructions, or with these steps:

On macOS:

```
# Download the Miniconda installer to ~/Downloads/miniconda.sh
curl -L https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh -o ~/
↪Downloads/miniconda.sh
# Install Miniconda
bash ~/Downloads/miniconda.sh

# Follow the prompts

# Close and reopen your shell
```

On GNU/Linux:

```
# Download the Miniconda installer to ~/miniconda.sh
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/
↪miniconda.sh
# Install Miniconda
bash ~/miniconda.sh

# Follow the prompts

# Close and reopen your shell
```

Note: on some Linux distributions (including Ubuntu and its derivatives), the csh shell that comes with the system is not compatible with conda. You will need to install the tcsh shell (which is backwards compatible with csh), and configure your system to use tcsh as csh:

```
# Install tcsh
sudo apt-get install tcsh
# Configure your system to use tcsh as csh
sudo update-alternatives --set csh /bin/tcsh
```

Initializing your shell for use with conda

We recommend initializing your default shell to use conda. This process has to be done only once.

The Miniconda installer should ask you if you want to do that as part of the installation procedure. If you did not answer "yes", you can use one of the following procedures depending on your default shell. Bash should be your default shell if you are on macOS (10.14 and older) or GNU/Linux.

Note: answering "yes" during the Miniconda installation procedure will only initialize the Bash shell for use with conda.

If your Mac has macOS 10.15 or higher, your default shell is Zsh.

These instructions make sure that the `conda` command is available when you start your shell by modifying your shell's startup file. Also, they make sure not to activate the "base" conda environment when you start your shell. This conda environment is created during the Miniconda installation but is not used for Icepack.

For Bash:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/bin/activate
conda init bash

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For Zsh (Z shell):

```
# Initialize Zsh to use conda
source $HOME/miniconda3/bin/activate
conda init zsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For tcsh:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/etc/profile.d/conda.csh
conda init tcsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For fish:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/etc/fish/conf.d/conda.fish
conda init fish
```

(continues on next page)

(continued from previous page)

```
# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For xonsh:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source-bash $HOME/miniconda3/bin/activate
conda init xonsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

Initializing your shell for conda manually

If you prefer not to modify your shell startup files, you will need to run the appropriate `source` command below (depending on your default shell) before using any conda command, and before compiling and running Icepack. These instructions make sure the `conda` command is available for the duration of your shell session.

For Bash and Zsh:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/bin/activate
```

For tcsh:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/etc/profile.d/conda.csh
```

For fish:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/etc/fish/conf.d/conda.fish
```

For xonsh:

```
# Initialize your shell session to use conda:
source-bash $HOME/miniconda3/bin/activate
```

Creating Icepack directories and the conda environment

The conda configuration expects some directories and files to be present at `$HOME/icepack-dirs`:

```
cd $HOME
mkdir -p icepack-dirs/runs icepack-dirs/baseline icepack-dirs/input
# Download the required forcing from https://github.com/CICE-Consortium/Icepack/wiki/
↪Icepack-Input-Data
# and untar it at $HOME/icepack-dirs/input
```

This step needs to be done only once.

If you prefer that some or all of the Icepack directories be located somewhere else, you can create a symlink from your home to another location:

```
# Create the Icepack directories at your preferred location
cd ${somewhere}
mkdir -p icepack-dirs/runs icepack-dirs/baseline icepack-dirs/input
# Download the required forcing from https://github.com/CICE-Consortium/Icepack/wiki/
↪Icepack-Input-Data
# and untar it at icepack-dirs/input

# Create a symlink to icepack-dirs in your $HOME
cd $HOME
ln -s ${somewhere}/icepack-dirs icepack-dirs
```

Note: if you wish, you can also create a complete machine port for your computer by leveraging the conda configuration as a starting point. See *Porting*.

Next, create the "icepack" conda environment from the `environment.yml` file in the Icepack source code repository. You will need to clone Icepack to run the following command:

```
conda env create -f configuration/scripts/machines/environment.yml
```

This step needs to be done only once. If you ever need to update the conda environment because the required packages change or packages are out of date, do

```
conda env update -f configuration/scripts/machines/environment.yml
```

Using the conda configuration

Follow the general instructions in *Overview*, using the `conda` machine name and `macos` or `linux` as compiler names.

On macOS:

```
./icepack.setup -m conda -e macos -c ~/icepack-dirs/cases/case1
cd ~/icepack-dirs/cases/case1
./icepack.build
./icepack.run
```

On GNU/Linux:

```
./icepack.setup -m conda -e linux -c ~/icepack-dirs/cases/case1
cd ~/icepack-dirs/cases/case1
./icepack.build
./icepack.run
```

A few notes about the conda configuration:

- This configuration always runs the model interactively, such that `./icepack.run` and `./icepack.submit` are the same.
- You should not update the packages in the `icepack` conda environment, nor install additional packages.
- It is not recommeded to run other test suites than `quick_suite` or `travis_suite` on a personal computer.

- The conda environment is automatically activated when compiling or running the model using the `./icepack.build` and `./icepack.run` scripts in the case directory. These scripts source the file `env.conda_{linux.macos}`, which calls `conda activate icepack`.
- The environment also contains the Sphinx package necessesary to build the HTML documentation. For this use case you must manually activate the environment:

```
cd doc
conda activate icepack
make html
# Open build/html/index.html in your browser
conda deactivate  # to deactivate the environment
```

3.2.4 Forcing data

The input data space is defined on a per machine basis by the `ICE_MACHINE_INPUTDATA` variable in the **env.[machine]** file. That file space is often shared among multiple users, and it can be desirable to consider using a common file space with group read and write permissions such that a set of users can update the inputdata area as new datasets are available.

The code is currently configured to run in standalone mode on a 4-cell grid using atmospheric data, available as detailed on the wiki. These data files are designed only for testing the code, not for use in production runs or as observational data. Please do not publish results based on these data sets. Module **configuration/driver/icedrv_forcing.F90** can be modified to change the forcing data.

Icepack requires near surface atmospheric data at a single point which are set in `forcing_nml` with the `atm_data_type` in the namelist (see *Table of Icepack Settings*). The required fields to force icepack include: downwelling long wave and shortwave radiative fluxes, latent and sensible heat fluxes, precipitation rate, and near surface potential temperature and specific humidity. The filenames `atm_data_file`, `ocn_data_file`, `ice_data_file`, and `bgc_data_file` must also be provided for options other than the default and climatological forcing cases. Current filenames can be found in the options scripts in **configuration/scripts/options** and in the forcing data directories.

1) **Climate Forecast System (CFS)**

   Hourly atmospheric forcing from the National Centers for Environmental Prediction's (NCEP) Climate Forecast System, version 2 (CFSv2) [59] were utilized to generate a one-year time series for Icepack testing. These data were used to create the annual cycle at a point in the Beaufort Sea (70N, 220W) for the period of January 1 00:00UTC - December 31 23:00UTC, 2015. Additional locations can be provided for both hemispheres for the period of 1999-2015 for future testing. This dataset can be used to run for several years to reach equilibrium of the annual cycle.

   Atmospheric forcing fields consist of 2-m air temperature (K), specific humidity (kg/kg), 10-m wind velocity in the x and y directions (m/s), downward solar radiation (W/m^2), downward longwave radiation (W/m^2), and precipitation ($kg/m^2/s$). Icepack's boundary layer calculation is used to derive sensible and latent heat fluxes. In the namelist, set `atm_data_type = CFS` to use CFS atmospheric forcing.

2) **Field campaign derived**

   a) **Norwegian Young Sea Ice cruise (N-ICE)**

      Atmospheric, oceanic, and biogeochemical forcing are available from the 2015 Norwegian Young Sea Ice Cruise (N-ICE) [11]. These data are available daily, except for incoming atmospheric radiative forcing, which are available 6-hourly. The data correspond to the Arctic Ocean north of Svalbard along the N-ICE drift track (83N, 16E to 80N, 5E) from April 24, 2015 to June 6, 2015.

      Atmospheric forcing fields from [11] consist of 2-m air temperature (K), 2-m specific humidity (kg/kg), 10-m wind velocity in the x and y directions (m/s), downward solar radiation (W/m^2), and precipitation ($kg/m^2/s$). Icepack's boundary layer calculation is used to derive sensible and latent heat fluxes. In the namelist, set `atm_data_type = NICE` to use N-ICE atmospheric forcing.

Oceanic forcing fields are available from a Parallel Ocean Program (POP) 1-degree (gx1v3) simulation [9]. These fields consist of sea surface temperature (K), sea surface salinity (ppt), boundary layer depth (m), ocean velocity in the x and y direction (m/s), and deep ocean heat flux (W/m^2). In the namelist, set `ocn_data_type = NICE` to use N-ICE oceanic forcing.

Biogeochemical forcing fields are available from the World Ocean Atlas [19]. The biogeochemical fields provided are nitrate concentration ($mmol/m^3$) and silicate concentration ($mmol/m^3$). In the namelist, set `bgc_data_type = NICE` to use N-ICE biogeochemical forcing.

b) **Ice Station Polarstern (ISPOL)**

Atmospheric, oceanic, and biogeochemical forcing are available from the 2004 Ice Station Polarstern (ISPOL) [28]. These data can be used with both [6] and mushy layer thermodynamics. These data are available daily, except for incoming atmospheric radiative forcing, which are available 6-hourly. The data correspond to the Weddell Sea (67.9S, 54W) from June 16, 2004 to December 31, 2004.

Atmospheric forcing fields from [28] consist of 2-m air temperature (K), 2-m specific humidity (kg/kg), 10-m wind velocity in the x and y directions (m/s), downward solar radiation (W/m^2), and precipitation ($kg/m^2/s$). Icepack's boundary layer calculation is used to derive sensible and latent heat fluxes. In the namelist, set `atm_data_type = ISPOL` to use ISPOL atmospheric forcing.

Oceanic forcing fields are available from [28] derived from a POP 1-degree (gx1v3 simulation) [9]. These consist of sea surface temperature (K), sea surface salinity (ppt), boundary layer depth (m), ocean velocity in the x and y direction (m/s), and deep ocean heat flux (W/m^2). In the namelist, set `ocn_data_type = ISPOL` to use ISPOL oceanic forcing.

Biogeochemical forcing fields are available from the World Ocean Atlas [19]. The biogeochemical fields provided are nitrate concentration ($mmol/m^3$) and silicate concentration ($mmol/m^3$). In the namelist, set `bgc_data_type = ISPOL` to use ISPOL biogeochemical forcing.

c) **Surface HEat Budget of the Arctic (SHEBA)**

The ice opening and closing rates (1/s) are derived from the SHEBA data and have been used previously in Cecilia Bitz's column model. For additional information see the following websites:

- https://atmos.washington.edu/~bitz/column_model/
- https://atmos.washington.edu/~bitz/column_model/notes_forcing_data

At present, only the opening and closing rates (1/s) are used from the forcing data. In the namelist, set `ocn_data_type = SHEBA` to use this forcing in Icepack.

3) **Climatological** - Maykut and Untersteiner 1971 [45]

The climatological forcing consists of a monthly climatology of downward radiative fluxes, air temperature, relative humidity and wind speed compiled from Arctic ice station observations shown in Table 1 from [36]. Icepack's boundary layer calculation is used to derive sensible and latent heat fluxes. The snowfall follows the idealized specification used by [61] . To adjust the ice thickness a fixed heating of 6 W/m^2 is applied to the bottom of the ice. This may be thought of as containing about 2 W/m^2 of ocean heating and an adjustment of about 4 W/m^2 for biases in the forcings or the model. In the namelist, set `atm_data_type = clim` to use climatological atmospheric forcing.

3.2.5 Horizontal ice advection

When Icepack is run in standalone mode with a dynamical forcing (e.g., `ocn_data_type = SHEBA`), closing implies the lateral flux of ice or open water area into the grid cell. The default assumption (in the namelist, `lateral_flux_type = 'uniform_ice'`) is that the active grid cell is surrounded by grid cells with identical ice properties to the active grid cell, i.e. the ice is uniform across all of those cells, and when the dynamical forcing is net convergence, this uniform ice is fluxed into the grid cell. Alternatively, one may assume that the active grid cell is surrounded by open water (in the namelist `lateral_flux_type = 'open_water'`), in which case closing (i.e., ice convergence) will produce open water in the grid cell. In either case, when the forcing is net divergence, ice area and volume are removed from the grid cell to accommodate the formation of open water implied by the net divergence.

3.2.6 Run Directories

The **icepack.setup** script creates a case directory. However, the model is actually built and run under the `ICE_OBJDIR` and `ICE_RUNDIR` directories as defined in the **icepack.settings** file. It's important to note that when the run script is submitted, the current **icepack_in**, **icepack.settings**, and **env.[machine]** files are copied from the case directory into the run directory. Users should generally not edit files in the run directory as these are overwritten when following the standard workflow.

Build and run logs will be copied from the run directory into the case **logs/** directory when complete.

3.2.7 Local modifications

Scripts and other case settings can be changed manually in the case directory and used. Source code can be modified in the main sandbox. When changes are made, the code should be rebuilt before being resubmitted. It is always recommended that users modify the scripts and input settings in the case directory, NOT the run directory. In general, files in the run directory are overwritten by versions in the case directory when the model is built, submitted, and run.

3.3 Testing Icepack

This section documents primarily how to use the Icepack scripts to carry out icepack testing. Exactly what to test is a separate question and depends on the kinds of code changes being made. Prior to merging changes to the CICE Consortium main branch, changes will be reviewed and developers will need to provide a summary of the tests carried out.

There is a base suite of tests provided by default with Icepack and this may be a good starting point for testing.

The testing scripts support several features

- Ability to test individual (via `--test`)or multiple tests (via `--suite`) using an input file to define the suite or suites
- Ability to use test suite defined in the package or test suites defined by the user
- Ability to store test results for regresssion testing (`--bgen`)
- Ability to compare results to prior baselines to verify bit-for-bit (`--bcmp`)
- Ability to define where baseline tests are stored
- Ability to compare tests against each other (`--diff`)

3.3.1 Individual Tests

The Icepack scripts support both setup of individual tests as well as test suites. Individual tests are run from the command line:

```
./icepack.setup --test smoke --mach conrad --env cray --set diag1,debug --testid myid
```

Tests are just like cases but have some additional scripting around them. Individual tests can be created and manually modified just like cases. Many of the command line arguments for individual tests are similar to *Command Line Options* for `--case`. For individual tests, the following command line options can be set

`--test` TESTNAME
: specifies the test type. This is probably either smoke or restart but see *icepack.setup –help* for the latest. This is required instead of `--case`.

`--testid` ID
: specifies the testid. This is required for every use of `--test` and `--suite`. This is a user defined string that will allow each test to have a unique case and run directory name. This is also required.

`--tdir` PATH
: specifies the test directory. Testcases will be created in this directory. (default is .)

`--mach` MACHINE (see *Command Line Options*)

`--env` ENVIRONMENT1 (see *Command Line Options*)

`--set` SET1,SET2,SET3 (see *Command Line Options*)

`--acct` ACCOUNT (see *Command Line Options*)

`--grid` GRID (see *Command Line Options*)

`--pes` MxN (see *Command Line Options*)

Like `--case`, `--grid` and `--pes` are not particularly useful right now within Icepack since the model can only run serially and only with the col grid setting. There are several additional options that come with `--test` that are not available with `--case` for regression and comparision testing,

`--bdir` DIR
: specifies the top level location of the baseline results. This is used in conjuction with `--bgen` and `--bcmp`. The default is set by ICE_MACHINE_BASELINE in the env.[machine]_[environment] file.

`--bgen` DIR
: specifies the name of the directory under [bdir] where test results will be stored. When this flag is set, it automatically creates that directory and stores results from the test under that directory. If DIR is set to `default`, then the scripts will automatically generate a directory name based on the Icepack hash and the date and time. This can be useful for tracking the baselines by hash.

`--bcmp` DIR
: specifies the name of the directory under [bdir] that the current tests will be compared to. When this flag is set, it automatically invokes regression testing and compares results from the current test to those prior results. If DIR is set to `default`, then the script will automatically generate the last directory name in the [bdir] directory. This can be useful for automated regression testing.

`--diff` LONG_TESTNAME
: invokes a comparison against another local test. This allows different tests to be compared to each other. The restrictions are that the test has to already be completed and the testid has to match.

The format of the case directory name for a test will always be `[machine]_[env]_[test]_[grid]_[pes]_[sets].[testid]` The [sets] will always be sorted alphabetically by the script so `--set debug,diag1` and `--set diag1,debug` produces the same testname and test with _debug_diag1 in that order.

To build and run a test, the process is the same as a case. cd to the test directory, run the build script, and run the submit script:

```
cd [test_case]
./icepack.build
./icepack.submit
```

The test results will be generated in a local file called **test_output**. To check those results:

```
cat test_output
```

Tests are defined under **configuration/scripts/tests/**. The tests currently supported are:

- **smoke - Runs the model for default length. The length and options can**
  be set with the `--set` command line option. The test passes if the model completes successfully.
- **restart - Runs the model for 14 months, writing a restart file at month 3 and**
  again at the end of the run. Runs the model a second time starting from the month 3 restart and writing a restart at month 12 of the model run. The test passes if both runs complete and if the restart files at month 12 from both runs are bit-for-bit identical.

Please run `./icepack.setup --help` for the latest information.

Individual Test Examples

1) **Basic default single test**

   Define the test, mach, env, and testid.

```
./icepack.setup --test smoke --mach wolf --env gnu --testid t00
cd wolf_gnu_smoke_col_1x1.t00
./icepack.build
./icepack.submit
./cat test_output
```

2) **Simple test with some options**

   Add `--set`

```
./icepack.setup --test smoke --mach wolf --env gnu --set diag1,debug --testid t00
cd wolf_gnu_smoke_col_1x1_debug_diag1.t00
./icepack.build
./icepack.submit
./cat test_output
```

3) **Single test, generate baseline dataset**

   Add `--bgen`

```
./icepack.setup --test smoke --mach wolf -env gnu --bgen icepack.v01 --testid t00 --
↪set diag1
cd wolf_gnu_smoke_col_1x1_diag1.t00
./icepack.build
./icepack.submit
./cat test_output
```

4) **Single test, compare results to a prior baseline**

   Add `--bcmp`. For this to work, the prior baseline must exist and have the exact same base testname [machine]_[env]_[test]_[grid]_[pes]_[sets]

```
./icepack.setup --test smoke --mach wolf -env gnu --bcmp icepack.v01 --testid t01 --
↪set diag1
cd wolf_gnu_smoke_col_1x1_diag1.t01
./icepack.build
./icepack.submit
./cat test_output
```

5) **Simple test, generate a baseline dataset and compare to a prior baseline**

   Use `--bgen` and `--bcmp`. The prior baseline must exist already.

```
./icepack.setup --test smoke --mach wolf -env gnu --bgen icepack.v02 --bcmp icepack.
↪v01 --testid t02 --set diag1
cd wolf_gnu_smoke_col_1x1_diag1.t02
./icepack.build
./icepack.submit
./cat test_output
```

6) **Simple test, comparison against another test**

   Use `--diff`. This feature is primarily used in test suites and has limited use in icepack, but is being described for completeness.

   `--diff` provides a way to compare tests with each other. For this to work, the tests have to be run in a specific order and the testids need to match. The test is always compared relative to the current case directory.

   To run the first test,

```
./icepack.setup --test smoke --mach wolf -env gnu --testid tx01 --set debug
cd wolf_gnu_smoke_col_1x1_debug.tx01
./icepack.build
./icepack.submit
./cat test_output
```

   Then to run the second test and compare to the results from the first test

```
./icepack.setup --test smoke --mach wolf -env gnu --testid tx01 --diff smoke_col_
↪1x1_debug
cd wolf_gnu_smoke_col_1x1.tx01
./icepack.build
./icepack.submit
./cat test_output
```

   The scripts will add a [machine]_[environment] to the beginning of the diff argument and the same testid to the end of the diff argument. Then the runs will be compared for bit-for-bit and a result will be produced in test_output. This is really more useful in CICE and for test suites right now. For example, CICE uses this feature to compare results from different pe counts or decompositions, single threaded vs multi-threaded, and so forth.

3.3.2 Test suites

Test suites support running multiple tests specified via an input file or files. When invoking the test suite option (`--suite`) with **icepack.setup**, all tests will be created, built, and submitted automatically under a directory called testsuite.[testid].[$date] as part of involing the suite. The test scripts provide an ability to reuse binaries between tests within a test suite when possible. Because the tests are built and submitted automatically, this feature does not allow for customization of cases or tests like individual cases and tests do:

```
./icepack.setup --suite base_suite --mach wolf --env gnu --testid myid
```

Like an individual test, the `--testid` option must be specified and can be any string.

If using the `--tdir` option, that directory must not exist before the script is run. The tdir directory will be created by the script and it will be populated by all tests as well as scripts that support the test suite:

```
./icepack.setup --suite base_suite --mach wolf --env gnu --testid myid --tdir /scratch/
↪$user/testsuite.myid
```

Once the tests are complete, results can be checked by running the results.csh script in the [suite_name].[testid]:

```
cd testsuite.[testid]
./results.csh
```

The predefined test suites are defined under **configuration/scripts/tests** and the files defining the suites have a suffix of .ts in that directory. The format for the test suite file is relatively simple. It is a text file with white space delimited columns that define a handful of values in a specific order. The first column is the test name, the second the grid, the third the pe count, the fourth column is the `--set` options and the fifth column is the `--diff` argument. (The grid and PEs columns are provided for compatibility with the similar CICE scripts.) The fourth and fifth columns are optional. Lines that begin with # or are blank are ignored. For example,

```
#Test   Grid  PEs  Sets             Diff
 smoke   col  1x1  diag1
 smoke   col  1x1  diag1,run1year  smoke_col_1x1_diag1
 smoke   col  1x1  debug,run1year
restart  col  1x1  debug
restart  col  1x1  diag1
restart  col  1x1  pondlvl
restart  col  1x1  pondtopo
```

The argument to `--suite` defines the test suite (.ts) filename or filenames and that argument can contain a path. **icepack.setup** will look for the filename in the local directory, in **configuration/scripts/tests/**, or in the path defined by the `--suite` option.

Because many of the command line options are specified in the input file, ONLY the following options are valid for suites,

`--suite` suitename1,suitename2
  required, input filename with comma delimited list of suite or suites

`--mach` MACHINE
  required

`--env` ENVIRONMENT1,ENVIRONMENT2
  strongly recommended

`--acct` ACCOUNT
  optional

`--tdir` PATH
optional

`--testid` ID
required

`--bdir` DIR
optional, top level baselines directory and defined by default by ICE_MACHINE_BASELINE in **env.[machine]_[environment]**.

`--bgen` DIR
recommended, test output is copied to this directory under [bdir]

`--bcmp` DIR
recommended, test output are compared to prior results in this directory under [bdir]

`--report`
This is only used by `--suite` and when set, invokes a script that sends the test results to the results page when all tests are complete. Please see *Test Reporting* for more information.

`--coverage`
When invoked, code coverage diagnostics are generated. This will modify the build and reduce optimization and generate coverage reports using lcov or codecov tools. General use is not recommended, this is mainly used as a diagnostic to periodically assess test coverage. Please see *Code Coverage Testing* for more information.

Please see *Command Line Options* and *Individual Tests* for more details about how these options are used.

Test Suite Examples

1) **Basic test suite**

   Specify suite, mach, env, testid.

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a
cd base_suite.v01a
#wait for runs to complete
./results.csh
```

2) **Basic test suite with user defined test directory**

   Specify suite, mach, env, testid.

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a --tdir /
↪scratch/$user/ts.v01a
cd /scratch/$user/ts.v01a
#wait for runs to complete
./results.csh
```

3) **Multiple test suites on multiple environments**

   Specify multiple envs.

```
./icepack.setup --suite base_suite,quick_suite --mach conrad --env cray,pgi,
↪intel,gnu --testid v01a
cd testsuite.v01a
#wait for runs to complete
./results.csh
```

The interface supports both multiple suites and multiple environments from a single command line invokation. Each env or suite can also be run as a separate invokation of *icepack.setup* but if that approach is taken, it is recommended that different testids be used.

4) **Basic test suite, store baselines in user defined name**

   Add `--bgen`

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen icepack.v01a
cd testsuite.v01a
#wait for runs to complete
./results.csh
```

   This will store the results in the default [bdir] directory under the subdirectory icepack.v01a.

5) **Basic test suite, store baselines in user defined top level directory**

   Add `--bgen` and `--bdir`

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen icepack.v01a --bdir /tmp/user/ICEPACK_BASELINES
cd testsuite.v01a
#wait for runs to complete
./results.csh
```

   This will store the results in /tmp/user/ICEPACK_BASELINES/icepack.v01a.

6) **Basic test suite, store baselines in auto-generated directory**

   Add `--bgen default`

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen default
cd testsuite.v01a
#wait for runs to complete
./results.csh
```

   This will store the results in the default [bdir] directory under a directory name generated by the script that includes the hash and date.

7) **Basic test suite, compare to prior baselines**

   Add `--bcmp`

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v02a --
↪bcmp icepack.v01a
cd testsuite.v02a
#wait for runs to complete
./results.csh
```

   This will compare to results saved in the baseline [bdir] directory under the subdirectory icepack.v01a. You can use other regression options as well (`--bdir` and `--bgen`)

8) **Basic test suite, use of default string in regression testing**

   default is a special argument to `--bgen` and `--bcmp`. When used, the scripts will automate generation of the directories. In the case of `--bgen`, a unique directory name consisting of the hash and a date will be created. In the case of `--bcmp`, the latest directory in [bdir] will automatically be specified. This provides a number of useful features

- the `--bgen` directory will be named after the hash automatically
- the `--bcmp` will always find the most recent set of baselines
- the `--bcmp` reporting will include information about the comparison directory name which will include hash information
- automation can be invoked easily, especially if `--bdir` is used to separate results

Imagine the case where the default settings are used and `--bdir` is used to create a unique location. You could easily carry out regular builds automatically via,

```
set mydate = `date -u "+%Y%m%d"`
git clone https://github.com/myfork/icepack icepack.$mydate
cd icepack.$mydate
./icepack.setup --suite base_suite --mach conrad --env cray,gnu,intel,pgi --
↪testid $mydate --bcmp default --bgen default --bdir /tmp/work/user/
↪ICEPACK_BASELINES_MAIN
```

When this is invoked, a new set of baselines will be generated and compared to the prior results each time without having to change the arguments.

9) **Create and test a custom suite**

   Create your own input text file consisting of 5 columns of data,

   - Test
   - Grid
   - pes
   - sets (optional)
   - diff test (optional)

   such as

```
> cat mysuite
smoke    col  1x1  diag1,debug
restart  col  1x1
restart  col  1x1  diag1,debug    restart_col_1x1
restart  col  1x1  mynewoption,diag1,debug
```

   then use that input file, mysuite

```
./icepack.setup --suite mysuite --mach conrad --env cray --testid v01a --
↪bgen default
cd mysuite.v01a
#wait for runs to complete
./results.csh
```

   You can use all the standard regression testing options (`--bgen`, `--bcmp`, `--bdir`). Make sure any "diff" testing that goes on is on tests that are created earlier in the test list, as early as possible. Unfortunately, there is still no absolute guarantee the tests will be completed in the correct sequence.

3.3.3 Test Reporting

The Icepack testing scripts have the capability to post test results to the official wiki page. You may need write permission on the wiki. If you are interested in using the wiki, please contact the consortium.

To post results, once a test suite is complete, run `results.csh` and `report_results.csh` from the suite directory,

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a
cd testsuite.v01a
#wait for runs to complete
./results.csh
./report_results.csh
```

`report_results.csh` will run `results.csh` by default automatically, but we recommmend running it manually first to verify results before publishing them. `report_results.csh -n` will turn off automatic running of `results.csh`.

The reporting can also be automated in a test suite by adding `--report` to `icepack.setup`

```
./icepack.setup --suite base_suite --mach conrad --env cray --testid v01a --report
```

With `--report`, the suite will create all the tests, build and submit them, wait for all runs to be complete, and run the results and report_results scripts.

3.3.4 Code Coverage Testing

The `--coverage` feature in **icepack.setup** provides a method to diagnose code coverage. This argument turns on special compiler flags including reduced optimization and then invokes the gcov tool. Once runs are complete, either lcov or codecov can be used to analyze the results. This option is currently only available with the gnu compiler and on a few systems with modified Macros files.

At the present time, the `--coverage` flag invokes the lcov analysis automatically by running the **report_lcov.csh** script in the test suite directory. The output will show up at the CICE-Consortium code coverage website. To use the tool, you should have write permission for that repository. The lcov tool should be run on a full multi-suite test suite, and it can take several hours to process the data once the test runs are complete. A typical instantiation would be

```
./icepack.setup --suite base_suite,travis_suite,quick_suite --mach cheyenne --env gnu --
↪testid cc01 --coverage
```

Alternatively, codecov analysis can be carried out by manually running the **report_codecov.csh** script from the test suite directory, but there are several ongoing problems with this approach and it is not generally recommended. The codecov analysis is largely identical to the analysis performed by lcov, codecov just provides a nicer web experience to view the output.

This is a special diagnostic test and is not part of the standard model testing. General use is not recommended, this is mainly used as a diagnostic to periodically assess test coverage.

3.3.5 Test Plotting

The Icepack scripts include a script (`timeseries.csh`) that will generate a timeseries figure from the diagnostic output file. When running a test suite, the `timeseries.csh` script is automatically copied to the suite directory. If the `timeseries.csh` script is to be used on a test / case that is not a part of a test suite, users will need to run the `timeseries.csh` script from the tests directory (`./configuration/scripts/tests/timeseries.csh`), or copy it to a local directory and run it locally (`cp configuration/scripts/tests/timeseries.csh .` followed by `./timeseries.csh /path/to/ice_diag.full_ITD`. The plotting script can be run on any of the output files - icefree, slab, full_ITD, land). To generate the figure, run the `timeseries.csh` script and pass the full path to the ice_diag file as an argument.

For example:

Run the test suite.

```
$ ./icepack.setup -m conrad -e intel --suite base_suite -acct <account_number> --testid␣
↪t00
```

Wait for suite to finish then go to the directory.

```
$ cd testsuite.t00
```

Run the timeseries script on the desired case.

```
$ ./timeseries.csh /p/work1/turner/ICEPACK_RUNS/conrad_intel_smoke_col_1x1_diag1_
↪run1year.t00/ice_diag.full_ITD
```

The output figures are placed in the directory where the ice_diag file is located.

This plotting script can be used to plot the following variables:

- area fraction
- average ice thickness (m)
- average snow depth (m)
- air temperature (C)
- shortwave radiation (W/m^2)
- longwave radiation (W/m^2)
- snowfall
- average salinity (ppt)
- surface temperature (C)
- outward longwave flux (W/m^2)
- sensible heat flux (W/m^2)
- latent heat flux (W/m^2)
- top melt (m)
- bottom melt (m)
- lateral melt (m)
- new ice (m)
- congelation (m)

- snow-ice (m)
- initial energy change (W/m^2)

3.4 Case Settings, Model Namelist, and CPPs

There are two important files that define the case, **icepack.settings** and **icepack_in**. **icepack.settings** is a list of env variables that define many values used to setup, build and run the case. **icepack_in** is the input namelist file for the icepack driver. Variables in both files are described below. In addition, the first table documents available C Preprocessor Macros.

3.4.1 Table of C Preprocessor (CPP) Macros

The Icepack model supports a few C Preprocessor (CPP) Macros. These can be turned on during compilation to activate different pieces of source code. The main purpose is to introduce build-time code modifications to include or exclude certain libraries or Fortran language features, in part to support CICE or other applications. More information can be found in *C Preprocessor (CPP) Macros*. The following CPPs are available.

Table 3: **CPP Macros**

CPP name	description
General Macros	
NO_I8	Converts `integer*8` to `integer*4`.
NO_R16	Converts `real*16` to `real*8`.
NO_SNICARHC	Does not compile hardcoded (HC) 5 band snicar tables tables needed by `shortwave=dEdd_snicar_ad`. May reduce compile time.
USE_NETCDF	Turns on netcdf capabilities in Icepack. By default and generally, Icepack does not need netcdf.
Application Macros	
CESMCOUPLED	Turns on code changes for the CESM coupled application
CICE_IN_NEMO	Turns on code changes for coupling in the NEMO ocean model

3.4.2 Table of Icepack Settings

The **icepack.settings** file contains a number of environment variables that define configuration, file system, run, and build settings. Several variables are set by the **icepack.setup** script. This file is created on a case by case basis and can be modified as needed.

Table 4: **Icepack settings**

variable	options/format	description	default value
ICE_CASENAME	string	case name	set by icepack.setup
ICE_SANDBOX	string	sandbox directory	set by icepack.setup
ICE_MACHINE	string	machine name	set by icepack.setup
ICE_ENVNAME	string	compilation environment	set by icepack.setup
ICE_MACHCOMP	string	machine_environment name	set by icepack.setup
ICE_SCRIPTS	string	scripts directory	set by icepack.setup

continues on next page

Table 4 – continued from previous page

variable	options/format	description	default value
ICE_CASEDIR	string	case directory	set by icepack.setup
ICE_RUNDIR	string	run directory	set by icepack.setup
ICE_OBJDIR	string	compile directory	${ICE_RUNDIR}/compile
ICE_RSTDIR	string	unused	${ICE_RUNDIR}/restart
ICE_HSTDIR	string	unused	${ICE_RUNDIR}/history
ICE_LOGDIR	string	log directory	${ICE_CASEDIR}/logs
ICE_RSTPFILE	string	unused	undefined
ICE_DRVOPT	string	unused	icepack
ICE_IOTYPE	none,netcdf	IO options	none
ICE_CLEANBUILD	true,false	automatically clean before building	true
ICE_CPPDEFS	string	user defined preprocessor macros for build	null
ICE_QUIETMODE	true, false	reduce build output to the screen	false
ICE_GRID	col	grid	col
ICE_NXGLOB	4	number of gridcells	4
ICE_NTASKS	1	number of tasks, must be set to 1	1
ICE_NTHRDS	1	number of threads per task, must be set to 1	1
ICE_TEST	string	test setting if using a test	set by icepack.setup
ICE_TESTNAME	string	test name if using a test	set by icepack.setup
ICE_BASELINE	string	baseline directory name, associated with icepack.setup -bd	set by icepack.setup
ICE_BASEGEN	string	baseline directory name for regression generation, associated with icepack.setup -bg	set by icepack.setup
ICE_BASECOM	string	baseline directory name for regression comparison, associated with icepack.setup -bc	set by icepack.setup
ICE_BFBCOMP	string	location of case for comparison, associated with icepack.setup -td	set by icepack.setup
ICE_SPVAL	string	unused	UnDeFiNeD
ICE_RUNLENGTH	string	batch run length default	00:10:00
ICE_ACCOUNT	string	batch account number	set by icepack.setup or by default
ICE_QUEUE	string	batch queue name	set by icepack.setup or by default
ICE_THREADED	true,false	force threading in compile, will always compile threaded if NTHRDS is gt 1	false
NICELYR	integer	number of vertical layers in the ice	7
NSNWLYR	integer	number of vertical layers in the snow	1
NICECAT	integer	number of ice thickness categories	5
NFSDCAT	integer	number of floe size categories	12
TRAGE	0,1	ice age tracer	1
TRFY	0,1	first-year ice area tracer	1
TRLVL	0,1	deformed ice tracer	1
TRPND	0,1	melt pond tracer	1
NTRAERO	integer	number of aerosol tracers	1

continues on next page

Table 4 – continued from previous page

variable	options/format	description	default value
NTRISO	integer	number of water isotope tracers	1
TRBRI	0,1	brine height tracer	0
TRZS		DEPRECATED	
TRBGCS	0,1	skeletal layer tracer, needs TRBGCZ=0	0
TRBGCZ	0,1	zbgc tracers, needs TRBGCS=0 and TRBRI=1	0
NBGCLYR	integer	number of zbgc layers	1
TRZAERO	0-6	number of z aerosol tracers	0
TRALG	0,1,2,3	number of algal tracers	0
TRDOC	0,1,2,3	number of dissolved organic carbon	0
TRDIC	0,1	number of dissolved inorganic carbon	0
TRDON	0,1	number of dissolved organic nitrogen	0
TRFEP	0,1,2	number of particulate iron tracers	0
TRFED	0,1,2	number of dissolved iron tracers	0
ICE_SNICARHC	true,false	include hardcoded (HC) snicar tables	false
ICE_BLDDEBUG	true,false	turn on compile debug flags	false
ICE_COVERAGE	true,false	turn on code coverage flags	false

3.4.3 Tables of Namelist Options

The Icepack driver reads a namelist input file, **icepack_in**, consisting of several namelist groups. The tables below summarize the different groups and the variables in each group. The variables are organized alphabetically and the default values listed are the values defined in the source code. Those values will be used unless overridden by the Icepack namelist file, **icepack_in**. The source code default values as listed in the table are not necessarily the recommended production values.

setup_nml

Table 5: **setup_nml namelist options**

variable	options/format	description	default value
`conserv_check`	logical	check conservation	`.false.`
`cpl_bgc`	logical	couple bgc thru driver	`.false.`
`days_per_year`	integer	number of days in a model year	365
`diagfreq`	integer	frequency of diagnostic output in timesteps	24
`diag_file`	string	diagnostic output filename	'ice_diag'
`dumpfreq`	`d`	write restart every `dumpfreq_n` days	`y`
	`m`	write restart every `dumpfreq_n` months	
	`y`	write restart every `dumpfreq_n` years	
`dump_last`	true/false	write restart at end of run	false
`dt`	seconds	thermodynamics time step length	3600.
`history_format`	`cdf`	history file output in netcdf format	`none`
	`none`	no history output	
`ice_ic`	`default`	latitude and sst dependent initial condition	`default`
	`none`	no ice	
	'path/file'	restart file name	
`istep0`	integer	initial time step number	0
`ndtd`	integer	number of dynamics/advection/ridging/steps per thermo timestep	1
`npt`	integer	total number of time steps to take	99999
`restart`	logical	initialize using restart file	`.false.`
`restart_dir`	string	path to restart directory	'./'
`restart_file`	string	output file prefix for restart dump	'iced'
`restart_format`	`bin`	restart file output in binary format	`bin`
	`cdf`	restart file output in netcdf format	
`use_leap_years`	logical	include leap days	`.false.`
`year_init`	integer	the initial year if not using restart	0

grid_nml

Table 6: **grid_nml namelist options**

variable	options/format	description	default value
`kcatbound`	`-1`	single category formulation	1
	`0`	old formulation	
	`1`	new formulation with round numbers	
	`2`	WMO standard categories	
	`3`	asymptotic scheme	

tracer_nml

Table 7: **tracer_nml namelist options**

variable	options/format	description	default value
`tr_aero`	logical	aerosols	`.false.`
`tr_fsd`	logical	floe size distribution	`.false.`
`tr_FY`	logical	first-year ice area	`.false.`
`tr_iage`	logical	ice age	`.false.`
`tr_iso`	logical	isotopes	`.false.`
`tr_lvl`	logical	level ice area and volume	`.false.`
`tr_pond_lvl`	logical	level-ice melt ponds	`.false.`
`tr_pond_topo`	logical	topo melt ponds	`.false.`
`tr_snow`	logical	advanced snow physics	`.false.`

thermo_nml

Table 8: **thermo_nml namelist options**

variable	options/format	description	default value
`a_rapid_mode`	real	brine channel diameter in m	0.5e-3
`aspect_rapid_mode`	real	brine convection aspect ratio	1.0
`conduct`	`bubbly`	conductivity scheme [50]	`bubbly`
	`MU71`	conductivity [45]	
`dSdt_slow_mode`	real	slow drainage strength parameter m/s/K	-1.5e-7
`floediam`	real	effective floe diameter for lateral melt in m	300.0
`hfrazilmin`	real	min thickness of new frazil ice in m	0.05
`hi_min`	real	minimum ice thickness allowed for thermo in m	0.01
`kitd`	`0`	delta function ITD approximation	1
	`1`	linear remapping ITD approximation	
`ksno`	real	snow thermal conductivity	0.3
`ktherm`	`-1`	thermodynamic model disabled	1
	`1`	Bitz and Lipscomb thermodynamic model	
	`2`	mushy-layer thermodynamic model	
`phi_c_slow_mode`	$0 < \phi_c < 1$	critical liquid fraction	0.05
`phi_i_mushy`	$0 < \phi_i < 1$	solid fraction at lower boundary	0.85
`Rac_rapid_mode`	real	critical Rayleigh number	10.0
`Tliquidus_max`	real	maximum liquidus temperature of mush (C)	0.0
`tscale_pnd_drain`	real	mushy pond macroscopic drainage timescale in days	10.

dynamics_nml

Table 9: **dynamics_nml namelist options**

variable	options/format	description	default value
`Cf`	real	ratio of ridging work to PE change in ridging	17.0
`kstrength`	`0`	ice strength formulation [21]	1
	`1`	ice strength formulation [57]	
`krdg_partic`	`0`	old ridging participation function	1
	`1`	new ridging participation function	
`krdg_redist`	`0`	old ridging redistribution function	1
	`1`	new ridging redistribution function	
`mu_rdg`	real	e-folding scale of ridged ice for `krdg_partic` = 1 in m^0.5	3.0

shortwave_nml

Table 10: **shortwave_nml namelist options**

variable	options/format	description	default value
`ahmax`	real	albedo is constant above this thickness in meters	0.3
`albedo_type`	`ccsm3`	NCAR CCSM3 albedo implementation	`ccsm3`
	`constant`	four constant albedos	
`albicei`	$0 < \alpha < 1$	near infrared ice albedo for thicker ice	0.36
`albicev`	$0 < \alpha < 1$	visible ice albedo for thicker ice	0.78
`albsnowi`	$0 < \alpha < 1$	near infrared, cold snow albedo	0.70
`albsnowv`	$0 < \alpha < 1$	visible, cold snow albedo	0.98
`dT_mlt`	real	Δ temperature per Δ snow grain radius	1.5
`kalg`	real	absorption coefficient for algae	0.6
`rsnw_mlt`	real	maximum melting snow grain radius	1500.
`R_ice`	real	tuning parameter for sea ice albedo from Delta-Eddington shortwave	0.0
`R_pnd`	real	tuning parameter for ponded sea ice albedo from Delta-Eddington shortwave	0.0
`R_snw`	real	tuning parameter for snow (broadband albedo) from Delta-Eddington shortwave	1.5
`shortwave`	`ccsm3`	NCAR CCSM3 shortwave distribution method	`dEdd`
	`dEdd`	Delta-Eddington method (3-band)	
	`dEdd_snicar_ad`	Delta-Eddington method with 5-band snow	
`snw_ssp_table`	`snicar`	lookup table for *dEdd_snicar_ad*	
	`test`	reduced lookup table for *dEdd_snicar_ad* testing	
`sw_dtemp`	real	temperature from melt for sw_redist	0.02
`sw_frac`	real	fraction of shortwave redistribution	0.9
`sw_redist`	logical	shortwave redistribution	.false.

ponds_nml

Table 11: **ponds_nml namelist options**

variable	options/format	description	default value
`dpscale`	real	scaling factor for flushing in permeable ice (ktherm=1)	1.e-3
`frzpnd`	`cesm`	CESM pond refreezing forumulation	`cesm`
	`hlid`	Stefan refreezing with pond ice thickness	
`hp1`	real	critical ice lid thickness for topo ponds in m	0.01
`hs0`	real	snow depth of transition to bare sea ice in m	
`hs1`	real	snow depth of transition to pond ice in m	0.03
`pndaspect`	real	aspect ratio of pond changes (depth:area)	0.8
`rfracmax`	$0 \leq r_{max} \leq 1$	maximum melt water added to ponds	0.85
`rfracmin`	$0 \leq r_{min} \leq 1$	minimum melt water added to ponds	0.15

snow_nml

Table 12: **snow_nml namelist options**

variable	options/format	description	default value
`drhosdwind`	real	wind compaction factor for snow	27.3
`rhosmin`	real	minimum snow density	100.0
`rhosmax`	real	maximum snow density	450.0
`rhosnew`	real	new snow density	100.0
`rsnw_fall`	real	radius of new snow (um)	54.526
`rsnw_tmax`	real	maximum snow radius (um)	1500.0
`snw_aging_table`	test	snow aging lookup table	test
	snicar	(not available in Icepack)	
`snwgrain`	logical	snow grain metamorphosis	.true.
`snwlvlfac`	real	fraction increase in bulk snow redistribution	0.3
`snwredist`	`snwITDrdg`	snow redistribution using ITD/ridges	snwITDrdg
	`bulk`	bulk snow redistribution	
	`none`	no snow redistribution	
`use_smliq_pnd`	logical	use liquid in snow for ponds	.true.
`windmin`	real	minimum wind speed to compact snow	10.0

forcing_nml

Table 13: **forcing_nml namelist options**

variable	options/format	description	default value
`atmbndy`	string	bulk transfer coefficients	`similarity`
	`similarity`	stability-based boundary layer	
	`constant`	constant-based boundary layer	

continues on next page

Table 13 – continued from previous page

variable	options/format	description	default value
	mixed	stability-based, but constant for sensible+latent heatfluxes	
atmiter_conv	real	convergence criteria for ustar	0.0
atm_data_file	string	file containing atmospheric data	' '
atm_data_format	bin	read direct access binary forcing files	bin
atm_data_type	clim	monthly climatology (see *Forcing data*)	default
	CFS	CFS model output (see *Forcing data*)	
	default	constant values defined in the code	
	ISPOL	ISPOL experiment data (see *Forcing data*)	
	NICE	N-ICE experiment data (see *Forcing data*)	
bgc_data_file	string	file containing biogeochemistry data	' '
bgc_data_format	bin	read direct access binary forcing files	bin
bgc_data_type	clim	bgc climatological data	default
	default	constant values defined in the code	
	ncar	POP ocean forcing data	
calc_strair	.false.	read wind stress and speed from files	.true.
	.true.	calculate wind stress and speed	
calc_Tsfc	logical	calculate surface temperature	.true.
cpl_frazil	external	frazil water/salt fluxes are handled outside of Icepack	fresh_ice_correction
	fresh_ice_correction	correct fresh-ice frazil water/salt fluxes for mushy physics	
	internal	send full frazil water/salt fluxes for mushy physics	
data_dir	string	path to forcing data directory	' '
default_season	summer	forcing initial summer values	winter
	winter	forcing initial winter values	
emissivity	real	emissivity of snow and ice	0.985
fbot_xfer_type	Cdn_ocn	variabler ocean heat transfer coefficient scheme	constant
	constant	constant ocean heat transfer coefficient	
formdrag	logical	calculate form drag	.false.
fyear_init	integer	first year of atmospheric forcing data	1998
highfreq	logical	high-frequency atmo coupling	.false.
lateral_flux_type	uniform_ice	flux ice with identical properties into the cell when closing (Icepack only)	
	none	advect open water into the cell when closing (Icepack only)	
ice_data_file	string	file containing ice opening, closing data	' '
l_mpond_fresh	.false.	release pond water immediately to ocean	.false.
	true	retain (topo) pond water until ponds drain	
natmiter	integer	number of atmo boundary layer iterations	5
oceanmixed_ice	logical	active ocean mixed layer calculation	.false.
ocn_data_file	string	file containing ocean data	' '
ocn_data_format	bin	read direct access binary forcing files	bin
ocn_data_type	default	constant values defined in the code	default
	ISPOL	ISPOL experiment data (see *Forcing data*)	
	NICE	N-ICE experiment data (see *Forcing data*)	
	SHEBA	Opening/closing dataset from SHEBA	

continues on next page

Table 13 – continued from previous page

variable	options/format	description	default value
`precip_units`	`mks`	liquid precipitation data units	`mks`
	`mm_per_month`		
	`mm_per_sec`	(same as MKS units)	
	`m_per_sec`		
`restore_ocn`	logical	restore sst to data	`.false.`
`saltflux_option`	`constant`	salt flux is referenced to a constant salinity	`constant`
	`prognostic`	use actual sea ice bulk salinity in flux	
`tfrz_option`	`constant`	constant ocean freezing temperature (Tocnfrz)	`mushy`
	`linear_salt`	linear function of salinity (ktherm=1)	
	`minus1p8`	constant ocean freezing temperature ($-1.8°C$)	
	`mushy`	matches mushy-layer thermo (ktherm=2)	
`trestore`	integer	sst restoring time scale (days)	90
`update_ocn_f`	`.false.`	do not include frazil water/salt fluxes in ocn fluxes	`.false.`
	`true`	include frazil water/salt fluxes in ocn fluxes	
`ustar_min`	real	minimum value of ocean friction velocity in m/s	0.005
`wave_spec_type`	`constant`	wave data file is provided, sea surface height generated using constant phase (1 iteration of wave fracture)	`none`
	`none`	no wave data provided, no wave-ice interactions (not recommended when using the FSD)	
	`profile`	no wave data file is provided, use fixed dummy wave spectrum, for testing, sea surface height generated using constant phase (1 iteration of wave fracture)	
	`random`	wave data file is provided, sea surface height generated using random number (multiple iterations of wave fracture)	
`ycycle`	integer	number of years in forcing data cycle	1

zbgc_nml

Table 14: **zbgc_nml namelist options**

variable	options/format	description	default value
`algaltype_diatoms`	real	mobility type between stationary and mobile algal diatoms	0.0
`algaltype_phaeo`	real	mobility type between stationary and mobile algal phaeocystis	0.5
`algaltype_sp`	real	mobility type between stationary and mobile small plankton	0.5
`algal_vel`	real	[33]	1.11e-8
`alpha2max_low_diatoms`	real	light limitation diatoms 1/(W/m^2)	0.8

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
`alpha2max_low_phaeo`	real	light limitation phaeocystis 1/(W/m^2)	0.67
`alpha2max_low_sp`	real	light limitation small plankton 1/(W/m^2)	0.67
`ammoniumtype`	real	mobility type between stationary and mobile ammonium	1.0
`beta2max_diatoms`	real	light inhibition diatoms 1/(W/m^2)	0.18
`beta2max_phaeo`	real	light inhibition phaeocystis 1/(W/m^2)	0.01
`beta2max_sp`	real	light inhibition small plankton 1/(W/m^2)	0.0025
`bgc_data_type`	`clim`	bgc climatological data	`default`
	`default`	constant values defined in the code	
	`ncar`	POP ocean forcing data	
`bgc_flux_type`	`constant`	constant ice–ocean flux velocity	`Jin2006`
	`Jin2006`	ice–ocean flux velocity of [30]	
`chlabs_diatoms`	real	chl absorbtion diatoms 1/m/(mg/m^3)	0.03
`chlabs_phaeo`	real	chl absorbtion phaeocystis 1/m/(mg/m^3)	0.05
`chlabs_sp`	real	chl absorbtion small plankton 1/m/(mg/m^3)	0.01
`dEdd_algae`	logical		`.false.`
`dmspdtype`	real	mobility type between stationary and mobile dmspd	-1.0
`dmspptype`	real	mobility type between stationary and mobile dmspp	0.5
`doctype_l`	real	mobility type between stationary and mobile doc lipids	0.5
`doctype_s`	real	mobility type between stationary and mobile doc saccharids	0.5
`dontype_protein`	real	mobility type between stationary and mobile don proteins	0.5
`dustFe_sol`	real	solubility fraction	0.005
`fedtype_1`	real	mobility type between stationary and mobile fed lipids	0.5
`feptype_1`	real	mobility type between stationary and mobile fep lipids	0.5
`frazil_scav`	real	increase in initial bio bracer from ocean scavenging	1.0
`fr_dFe`	real	fraction of remineralized nitrogen in units of algal iron	0.3
`fr_graze_diatoms`	real	fraction grazed diatoms	0.01
`fr_graze_e`	real	fraction of assimilation excreted	0.5
`fr_graze_phaeo`	real	fraction grazed phaeocystis	0.1
`fr_graze_s`	real	fraction of grazing spilled or slopped	0.5
`fr_graze_sp`	real	fraction grazed small plankton	0.1
`fr_mort2min`	real	fractionation of mortality to Am	0.5
`fr_resp`	real	frac of algal growth lost due to respiration	0.05
`fr_resp_s`	real	DMSPd fraction of respiration loss as DMSPd	0.75
`fsal`	real	salinity limitation ppt	1.0
`F_abs_chl_diatoms`	real	scales absorbed radiation for dEdd chl diatoms	2.0
`F_abs_chl_phaeo`	real	scales absorbed radiation for dEdd chl phaeocystis	5.0

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
F_abs_chl_sp	real	scales absorbed radiation for dEdd small plankton	4.0
f_doc_l	real	fraction of mortality to DOC lipids	0.4
f_doc_s	real	fraction of mortality to DOC saccharides	0.4
f_don_Am_protein	real	fraction of remineralized DON to ammonium	0.25
f_don_protein	real	fraction of spilled grazing to proteins	0.6
f_exude_l	real	fraction of exudation to DOC lipids	1.0
f_exude_s	real	fraction of exudation to DOC saccharids	1.0
grid_o	real	z biology for bottom flux	5.0
grid_oS	real	DEPRECATED	
grow_Tdep_diatoms	real	temperature dependence growth diatoms per degC	0.06
grow_Tdep_phaeo	real	temperature dependence growth phaeocystis per degC	0.06
grow_Tdep_sp	real	temperature dependence growth small plankton per degC	0.06
humtype	real	mobility type between stationary and mobile hum	1.0
initbio_frac	real	fraction of ocean trcr concentration in bio tracers	1.0
K_Am_diatoms	real	ammonium half saturation diatoms mmol/m^3	0.3
K_Am_phaeo	real	ammonium half saturation phaeocystis mmol/m^3	0.3
K_Am_sp	real	ammonium half saturation small plankton mmol/m^3	0.3
k_bac_l	real	Bacterial degredation of DOC lipids per day	0.03
k_bac_s	real	Bacterial degredation of DOC saccharids per day	0.03
k_exude_diatoms	real	algal exudation diatoms per day	0.0
k_exude_phaeo	real	algal exudation phaeocystis per day	0.0
k_exude_sp	real	algal exudation small plankton per day	0.0
K_Fe_diatoms	real	iron half saturation diatoms nM	1.0
K_Fe_phaeo	real	iron half saturation phaeocystis nM	0.1
K_Fe_sp	real	iron half saturation small plankton nM	0.2
k_nitrif	real	nitrification rate per day	0.0
K_Nit_diatoms	real	nitrate half saturation diatoms mmol/m^3	1.0
K_Nit_phaeo	real	nitrate half saturation phaeocystis mmol/m^3	1.0
K_Nit_sp	real	nitrate half saturation small plankton mmol/m^3	1.0
K_Sil_diatoms	real	silicate half saturation diatoms mmol/m^3	4.0
K_Sil_phaeo	real	silicate half saturation phaeocystis mmol/m^3	0.0
K_Sil_sp	real	silicate half saturation small plankton mmol/m^3	0.0
kn_bac_protein	real	bacterial degradation of DON per day	0.03
l_sk	real	characteristic diffusive scale in m	7.0
l_skS	real	DEPRECATED	

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
max_dfe_doc1	real	max ratio of dFe to saccharides in the ice in nm Fe / muM C	0.2
max_loss	real	restrict uptake to percent of remaining value	0.9
modal_aero	logical	modal aersols	.false.
mort_pre_diatoms	real	mortality diatoms	0.007
mort_pre_phaeo	real	mortality phaeocystis	0.007
mort_pre_sp	real	mortality small plankton	0.007
mort_Tdep_diatoms	real	temperature dependence of mortality diatoms per degC	0.03
mort_Tdep_phaeo	real	temperature dependence of mortality phaeocystis per degC	0.03
mort_Tdep_sp	real	temperature dependence of mortality small plankton per degC	0.03
mu_max_diatoms	real	maximum growth rate diatoms per day	1.2
mu_max_phaeo	real	maximum growth rate phaeocystis per day	0.851
mu_max_sp	real	maximum growth rate small plankton per day	0.851
nitratetype	real	mobility type between stationary and mobile nitrate	-1.0
op_dep_min	real	light attenuates for optical depths exceeding min	0.1
phi_snow	real	snow porosity for brine height tracer	0.5
ratio_chl2N_diatoms	real	algal chl to N in mg/mmol diatoms	2.1
ratio_chl2N_phaeo	real	algal chl to N in mg/mmol phaeocystis	0.84
ratio_chl2N_sp	real	algal chl to N in mg/mmol small plankton	1.1
ratio_C2N_diatoms	real	algal C to N in mol/mol diatoms	7.0
ratio_C2N_phaeo	real	algal C to N in mol/mol phaeocystis	7.0
ratio_C2N_proteins	real	algal C to N in mol/mol proteins	7.0
ratio_C2N_sp	real	algal C to N in mol/mol small plankton	7.0
ratio_Fe2C_diatoms	real	algal Fe to C in umol/mol diatoms	0.0033
ratio_Fe2C_phaeo	real	algal Fe to C in umol/mol phaeocystis	1.0
ratio_Fe2C_sp	real	algal Fe to C in umol/mol small plankton	0.0033
ratio_Fe2N_diatoms	real	algal Fe to N in umol/mol diatoms	0.23
ratio_Fe2N_phaeo	real	algal Fe to N in umol/mol phaeocystis	0.7
ratio_Fe2N_sp	real	algal Fe to N in umol/mol small plankton	0.23
ratio_Fe2DOC_s	real	Fe to C of DON saccharids nmol/umol	1.0
ratio_Fe2DOC_l	real	Fe to C of DOC lipids nmol/umol	0.033
ratio_Fe2DON	real	Fe to C of DON nmol/umol	0.023
ratio_Si2N_diatoms	real	algal Si to N in mol/mol diatoms	1.8
ratio_Si2N_phaeo	real	algal Si to N in mol/mol phaeocystis	0.0
ratio_Si2N_sp	real	algal Si to N in mol/mol small plankton	0.0
ratio_S2N_diatoms	real	algal S to N in mol/mol diatoms	0.03
ratio_S2N_phaeo	real	algal S to N in mol/mol phaeocystis	0.03
ratio_S2N_sp	real	algal S to N in mol/mol small plankton	0.03
restore_bgc	logical	restore bgc to data	.false.
R_dFe2dust	real	g/g [66]	0.035
scale_bgc	logical		.false.
silicatetype	real	mobility type between stationary and mobile silicate	-1.0
skl_bgc	logical	biogeochemistry	.false.

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
solve_zbgc	logical		.false.
solve_zsal	logical	DEPRECATED	.false.
tau_max	real	long time mobile to stationary exchanges	1.73e-5
tau_min	real	rapid module to stationary exchanges	5200.
tr_bgc_Am	logical	ammonium tracer	.false.
tr_bgc_C	logical	algal carbon tracer	.false.
tr_bgc_chl	logical	algal chlorophyll tracer	.false.
tr_bgc_DMS	logical	DMS tracer	.false.
tr_bgc_DON	logical	DON tracer	.false.
tr_bgc_Fe	logical	iron tracer	.false.
tr_bgc_hum	logical		.false.
tr_bgc_Nit	logical		.false.
tr_bgc_PON	logical	PON tracer	.false.
tr_bgc_Sil	logical	silicate tracer	.false.
tr_brine	logical	brine height tracer	.false.
tr_zaero	logical	vertical aerosol tracers	.false.
t_iron_conv	real	desorption loss pFe to dFe in days	3065.
t_sk_conv	real	Stefels conversion time in days	3.0
t_sk_ox	real	DMS oxidation time in days	10.0
T_max	real	maximum temperature degC	0.0
y_sk_DMS	real	fraction conversion given high yield	0.5
zaerotype_bc1	real	mobility type between stationary and mobile zaero bc1	1.0
zaerotype_bc2	real	mobility type between stationary and mobile zaero bc2	1.0
zaerotype_dust1	real	mobility type between stationary and mobile zaero dust1	1.0
zaerotype_dust2	real	mobility type between stationary and mobile zaero dust2	1.0
zaerotype_dust3	real	mobility type between stationary and mobile zaero dust3	1.0
zaerotype_dust4	real	mobility type between stationary and mobile zaero dust4	1.0
z_tracers	logical		.false.

- = If Icepack is run stand-alone and wave_spec_type is not set to none, then a fixed wave spectrum is defined in the code to use for testing. As with other input data, this spectrum should not be used for production runs or publications.

3.4.4 BGC Tuning Parameters

Biogeochemical tuning parameters are specified as namelist options in **icepack_in**. Table *Biogeochemical Reaction Parameters* provides a list of parameters used in the reaction equations, their representation in the code, a short description of each and the default values. Please keep in mind that there has only been minimal tuning of the model.

Table 15: *Biogeochemical Reaction Parameters*

Text Variable	Variable in code	Description	Value	units
f_{graze}	fr_graze(1:3)	fraction of growth grazed	0, 0.1, 0.1	1
f_{res}	fr_resp	fraction of growth respired	0.05	1
l_{max}	max_loss	maximum tracer loss fraction	0.9	1
m_{pre}	mort_pre(1:3)	maximum mortality rate	0.007, 0.007, 0.007	day^{-1}
m_T	mort_Tdep(1:3)	mortality temperature decay	0.03, 0.03, 0.03	$^o\text{C}^{-1}$
T_{max}	T_max	maximum brine temperature	0	^{o}C
k_{nitr}	k_nitrif	nitrification rate	0	day^{-1}
f_{ng}	fr_graze_e	fraction of grazing excreted	0.5	1
f_{gs}	fr_graze_s	fraction of grazing spilled	0.5	1
f_{nm}	fr_mort2min	fraction of mortality to NH_4	0.5	1
f_{dg}	f_don	frac. spilled grazing to DON	0.6	1
k_{nb}	kn_bac [a]	bacterial degradation of DON	0.03	day^{-1}
f_{cg}	f_doc(1:3)	fraction of mortality to DOC	0.4, 0.4, 0.2	1
$R^c_{c:n}$	R_C2N(1:3)	algal carbon to nitrogen ratio	7.0, 7.0, 7.0	mol/mol
k_{cb}	k_bac1:3[a]	bacterial degradation of DOC	0.03, 0.03, 0.03	day^{-1}
τ_{fe}	t_iron_conv	conversion time pFe $\leftrightarrow$ dFe	3065.0	day
$r^{max}_{fed:doc}$	max_dfe_doc1	max ratio of dFe to saccharids	0.1852	nM Fe/μM C
f_{fa}	fr_dFe	fraction of remin. N to dFe	0.3	1
$R_{fe:n}$	R_Fe2N(1:3)	algal Fe to N ratio	0.023, 0.023, 0.7	mmol/mol
$R_{s:n}$	R_S2N(1:3)	algal S to N ratio	0.03, 0.03, 0.03	mol/mol
f_{sr}	fr_resp_s	resp. loss as DMSPd	0.75	1
τ_{dmsp}	t_sk_conv	Stefels rate	3.0	day
τ_{dms}	t_sk_ox	DMS oxidation rate	10.0	day
y_{dms}	y_sk_DMS	yield for DMS conversion	0.5	1

continues on next page

Table 15 – continued from previous page

Text Variable	Variable in code	Description	Value	units
K_{NO_3}	K_Nit(1:3)	NO_3 half saturation constant	1,1,1	mmol/m^3
K_{NH_4}	K_Am(1:3)	NH_4 half saturation constant	0.3, 0.3, 0.3	mmol/m^{-3}
K_{SiO_3}	K_Sil(1:3)	silicate half saturation constant	4.0, 0, 0	mmol/m^{-3}
K_{fed}	K_Fe(1:3)	iron half saturation constant	1.0, 0.2, 0.1	μmol/m^{-3}
op_{min}	op_dep_min	boundary for light attenuation	0.1	1
$chlabs$	chlabs(1:3)	light absorption length per chla conc.	0.03, 0.01, 0.05	$1/\mathrm{m}/(\mathrm{mg}/\mathrm{m}^3)$
α	alpha2max_low(1:3)	light limitation factor	0.25, 0.25, 0.25	m^2/W
β	beta2max(1:3)	light inhibition factor	0.018, 0.0025, 0.01	m^2/W
μ_{max}	mu_max(1:3)	maximum algal growth rate	1.44, 0.851, 0.851	day^{-1}
μ_T	grow_Tdep(1:3)	temperature growth factor	0.06, 0.06, 0.06	day^{-1}
f_{sal}	fsal	salinity growth factor	1	1
$R_{si:n}$	R_Si2N(1:3)	algal silicate to nitrogen	1.8, 0, 0	mol/mol

^{a} only (1:2) of DOC and DOC parameters have physical meaning

3.5 Troubleshooting

Check the FAQ: https://github.com/CICE-Consortium/Icepack/wiki

3.5.1 Initial setup

If there are problems, you can manually edit the env, Macros, and **icepack.run** files in the case directory until things are working properly. Then you can copy the env and Macros files back to **configuration/scripts/machines**.

- Changes made directly in the run directory, e.g. to the namelist file, will be overwritten if scripts in the case directory are run again later.
- If changes are needed in the **icepack.run.setup.csh** script, it must be manually modified.

3.5.2 Restarts

- Manual restart tests require the path to the restart file be included in `ice_in` in the namelist file.
- Ensure that `kcatbound` is the same as that used to create the restart file. Other configuration parameters, such as `NICELYR`, must also be consistent between runs.

3.5.3 Debugging hints

Icepack has a warning package (**/columnphysics/icepack_warnings.F90**) where icepack stores information not set in write routines. Details about the package can be found in *Error Messages and Aborts*. This package can be useful to detect an abort

A printing utility is available in the driver that can be helpful when debugging the code. Not all of these will work everywhere in the code, due to possible conflicts in module dependencies.

***conserv_check* = true (ice_in)**
  Diagnoses conservation in various icepack algorithms.

***debug_icepack* (configuration/driver/ice_diagnostics.F90)**
  A wrapper for *print_state* that is easily called from numerous points during initialization and the timestepping loop

***print_state* (configuration/driver/ice_diagnostics.F90)**
  Print the ice state and forcing fields for a given grid cell.

3.5.4 Known bugs and other issues

- With the old CCSM radiative scheme (`shortwave` = 'default' or 'ccsm3'), a sizable fraction (more than 10%) of the total shortwave radiation is absorbed at the surface but should be penetrating into the ice interior instead. This is due to use of the aggregated, effective albedo rather than the bare ice albedo when `snowpatch` < 1.
- The linear remapping algorithm for thickness is not monotonic for tracers.
- The form drag parameterization assumes a fixed ridge shape where both the macroscopic ridge porosity and the angle of repose are specified parameters. One high-resolution coupled model that uses the CICE column physics package has been unable to make this parameterization work in its current form. Development of a new, variational approach for ridging is underway that will generate ridge shapes differently from the current parameterization, and is expected to alleviate the reported problem ([54]).

3.5.5 Interpretation of albedos

The snow-and-ice albedo, `albsni`, and diagnostic albedos `albice`, `albsno`, and `albpnd` are merged over categories but not scaled (divided) by the total ice area. (This is a change from CICE v4.1 for `albsni`.) The latter three history variables represent completely bare or completely snow- or melt-pond-covered ice; that is, they do not take into account the snow or melt pond fraction (`albsni` does, as does the code itself during thermodyamic computations). This is to facilitate comparison with typical values in measurements or other albedo parameterizations. The melt pond albedo `albpnd` is only computed for the Delta-Eddington shortwave case.

With the Delta-Eddington parameterization, the albedo depends on the cosine of the zenith angle ($\cos\varphi$, `coszen`) and is one if the sun is below the horizon ($\cos\varphi < 0$). Thus, the albedos will be one in the dark, polar winter hemisphere. However, the time-averaged albedo fields will be high if a diurnal solar cycle is used, because values of one would be included in the average for half of each 24-hour period. To rectify this, a separate counter should be used for the averaging that is incremented only when $\cos\varphi > 0$. However, this is still a work in progress.

3.5.6 Interpretation of general results

Icepack releases are “functional releases” in the sense that the code runs, does not crash, passes various tests, and requires further work to establish its scientific validity. In general, users are not encouraged to use any of the CICE Consortium’s model configurations to obtain “scientific” results. The test configurations are useful for model development, but sea ice models must be evaluated from a physical standpoint in a couple system because simplified configurations do not necessarily represent what is actually happening in the fully coupled system that includes interactive ocean and atmosphere components.

3.5.7 Proliferating subprocess parameterizations

With the addition of several alternative parameterizations for sea ice processes, a number of subprocesses now appear in multiple parts of the code with differing descriptions. For instance, sea ice porosity and permeability, along with associated flushing and flooding, are calculated separately for mushy thermodynamics, topo and level-ice melt ponds, and for the brine height tracer, each employing its own equations. Likewise, the Bitz99 and mushy thermodynamics compute freeboard and snow–ice formation differently, and the topo and level-ice melt pond schemes both allow fresh ice to grow atop melt ponds, using slightly different formulations for Stefan freezing. These various process parameterizations will be compared and their subprocess descriptions possibly unified in the future.

CHAPTER

FOUR

USE IN OTHER MODELS

4.1 Overview

Icepack is a column physics package designed to be used in other broader sea ice models, such as CICE, SIS, or even in ocean models. Icepack includes options for simulating sea ice thermodynamics, mechanical redistribution (ridging) and associated area and thickness changes. In addition, the model supports a number of tracers, including thickness, enthalpy, ice age, first-year ice area, deformed ice area and volume, melt ponds, and biogeochemistry.

Icepack is called on a grid point by grid point basis. All data is passed in and out of the model via subroutine interfaces. Fortran "use" statements are not encouraged for accessing data inside the Icepack model.

Icepack does not generally contain any parallelization or I/O. The driver of Icepack is expected to support those features. Icepack can be called concurrently across multiple MPI tasks. Icepack should also be thread safe.

4.2 Protocols

This section describes a number of basic protocols for using Icepack in other models.

4.2.1 Access

Icepack provides several public interfaces. These are defined in **columnphysics/icepack_intfc.F90**. Icepack interfaces all contain the icepack_ prefix. Icepack interfaces follow a general design where data is passed in on a gridpoint by gridpoint basis, that data is updated and returned to the driver, and the data is not stored within Icepack. Additional information about the interfaces can be found in *Sequencing and Interfaces*.

Icepack interfaces can have long argument lists. These are documented in *Public Interfaces*. In some cases, arguments are required for optional features (i.e. biogeochemistry) even when that feature is turned off in Icepack. The Icepack development team continues to work towards having more optional arguments. If an argument is required for the interface but not needed, the driver will still have to pass a (dummy) variable thru the interface to meet the interface specification.

4.2.2 Initialization

The subroutine **icepack_configure** should be called before any other icepack interfaces are called. This subroutine initializes the abort flag and a few other important defaults. We recommend that call be implemented as:

```
call icepack_configure()  ! initialize icepack
call icepack_warnings_flush(nu_diag)
if (icepack_warnings_aborted()) call my_abort_method()
```

The 2nd and 3rd line above are described further in *Error Messages and Aborts*.

4.2.3 Error Messages and Aborts

Icepack does not generally handle I/O (file units), the parallel computing environment (MPI, etc), or model aborts. Icepack generates and buffers error messages that can be accessed by the driver. In addition, if Icepack fails, it will set an abort flag that can be queried by the driver. To best use those features, it's recommended that after every icepack interface call, the user add the following:

```
call icepack_warnings_flush(nu_diag)
if (icepack_warnings_aborted()) call my_abort_method()
```

icepack_warnings_flush is a public interface in icepack that writes any warning or error messages generated in icepack to the driver file unit number defined by nu_diag. The function **icepack_warnings_aborted** queries the internal icepack abort flag and returns true if icepack generated an abort error. my_abort_method represents a method in the driver that will abort the model cleanly.

In addition to writing Icepack messages thru the icepack_warnings_flush interface, there are also several methods in icepack that write general information to a file. The various **icepack_write_** interfaces accept a unit number provided by the driver and then document internal Icepack values.

4.2.4 Setting Internal Parameters

While Icepack does not generally store the model state, there are several Icepack interfaces that allow the driver to set various scientific and technical parameters internally in Icepack for later use. Those interfaces generally have a set (init), get (query) and write method that allows the driver to set, get, or write the values defined internally. Some parameters are required to be set by the driver and others take on defaults. The following table defines the available interfaces that fit into this category.

Table 1: *Init, Query, and Write Interfaces*

type	init	query	write	notes
orbital	icepack_init_orbit	icepack_query_orbit		orbital settings
parameters	icepack_init_parameters	icepack_query_parameters	icepack_write_parameters	scientific parameters
tracer flags	icepack_init_tracer_flags	icepack_query_tracer_flags	icepack_write_tracer_flags	tracer flags
tracer sizes		icepack_query_tracer_sizes	icepack_write_tracer_sizes	tracer counts and tracer maximum sizes
tracer indices	icepack_init_tracer_indices	icepack_query_tracer_indices	icepack_write_tracer_indices	tracer indexing in a broader tracer array

Many of these interfaces are related to tracers and in particular, tracer indexing in broader arrays. This is further explained in *Tracer Indexing*.

4.2.5 Tracer Indexing

Tracers are really just variables associated with the model state. Some of the tracers are prognostic, vary each timestep, and are updated in Icepack. Other tracers are just used by Icepack to evolve other tracers.

One of the most complicated aspects of the Icepack usage are managing tracers. Some tracers (i.e. Tsfc, qice, qsno) are required, while other tracers (i.e. FY or bgc tracers) are optional and used only when certain features are triggered. As a general rule, Icepack is aware of only a specific set of tracers and each tracer takes on multiple properties including counts, dependencies (*Tracers that depend on other tracers*), and indexing in a broader tracer array. The following table summarize the various types of tracers understood by Icepack and lists some of their properties. See also *Biogeochemical Tracers*.

Table 2: *Tracer Types and Properties*

name	status	optional flag	number	count	notes
Tsfc	required		1	1	ice/snow temperature
qice	required		1	nilyr	ice enthalpy
qsno	required		1	nslyr	snow enthalpy
sice	required		1	nilyr	ice bulk salinity
iage	optional	tr_iage	1	1	ice age
FY	optional	tr_FY	1	1	first year ice
alvl	optional	tr_lvl	1	1	level ice area fraction
vlvl	optional	tr_lvl	1	1	level ice area volume
apnd	optional	tr_pond	1	1	melt pond area fraction
hpnd	optional	tr_pond	1	1	melt pond depth
ipnd	optional	tr_pond	1	1	melt pond refrozen thickness
fsd	optional	tr_fsd	1	nfsd	floe size distribution
iso	optional	tr_iso	n_iso	2	water isotopes (snow, sea ice)
aero	optional	tr_aero	n_aero*	4	aerosols (snow SSL, snow below SSL, sea ice SSL, sea ice below S
fbri	optional	tr_brine	1	1	
bgc_N	optional	tr_bgc_N	n_algae	nblyr+3	nutrients
bgc_Nit	optional		1	nblyr+3	diatoms, phaeocystis, pico/small
bgc_DOC	optional	tr_bgc_DOC	n_doc	nblyr+3	dissolved organic carbon
bgc_DIC	optional		n_dic	nblyr+3	dissolved inorganic carbon
bgc_chl	optional		n_algae	nblyr+3	algal chlorophyll
bgc_Am	optional	tr_bgc_Am	1	nblyr+3	ammonia
bgc_Sil	optional	tr_bgc_Sil	1	nblyr+3	silicon
bgc_DMSPp	optional	tr_bgc_DMS	1	nblyr+3	
bgc_DMSPd	optional	tr_bgc_DMS	1	nblyr+3	
bgc_DMS	optional	tr_bgc_DMS	1	nblyr+3	
bgc_PON	optional	tr_bgc_PON	1	nblyr+3	zooplankton and detritus
bgc_DON	optional	tr_bgc_DON	n_don	nblyr+3	dissolved organic nitrogen
bgc_Fed	optional	tr_bgc_Fe	n_fed	nblyr+3	dissolved iron
bgc_Fep	optional	tr_bgc_Fe	n_fep	nblyr+3	particulate iron
bgc_hum	optional	tr_bgc_hum	1	nblyr+3	humic material
zaero	optional	tr_zaero	n_zaero	nblyr+3	bgc aerosols like black carbon
zbgc_frac	optional		1	nbtrcr	fraction of tracer in mobile phase

- NOTE the aero tracer indexing is a little more complicated depending which aero option is chosen.

The nt_ start index in a full tracer array is the start index associated with tracer relative to the number*count. The nlt_ start index in a bgc array is the start index associated with the tracer relative to the number only and it generally contains only bgc tracers.

Generally, tracers are passed into the Icepack interfaces by type where each type is a separate argument. There are some cases where an array of tracers is required and this is where the tracer indexing is particularly important. Below is a list of the various tracer indexing used

- nt_ references the tracer start index in a broader tracer array
- nlt_ references a bgc specific tracer start index for a different bgc array with different indexing from the nt_ indexing
- trcrn_depend/strata/etc defines dependency properties for tracers associated with the full array reference by nt_ indexing
- bio_index and bio_index_o is something else

In **icepack_aggregate**, the arguments *trcr_depend*, *trcr_base*, *n_trcr_strata*, and *nt_strata* are passed into the interface, and they provide information on dependencies between tracers. This information needs to be initialized in the driving code. In the bgc implementation, there are arrays *bio_index* and *bio_index_o* which also need to be initialized in the driving code and passed to Icepack.

4.3 Sequencing and Interfaces

4.3.1 Access to Interfaces

Icepack public parameters and interfaces as accessed via a single module in icepack, **icepack_intfc.F90**. The standard syntax to gain access to Icepack parameters and interfaces is through Fortran90 use. For example:

```
use icepack_intfc, only: icepack_warnings_flush
use icepack_intfc, only: icepack_warnings_aborted
use icepack_intfc, only: icepack_query_tracer_indices
use icepack_intfc, only: icepack_configure
```

The full suite of public parameters and interfaces is documented in *icepack_intfc.F90*.

4.3.2 Interfaces Module

Column physics data and subroutines are made public through the **icepack_intfc.F90** file. That file contains the entire list of data and subroutines needed to initialize, setup, and run the column physics package. That file points to other modules within the column physics where the interfaces are located.

Within **icepack_intfc.F90**, internal icepack kinds are defined via the icepack_kinds module:

```
use icepack_kinds, only: icepack_char_len  => char_len
use icepack_kinds, only: icepack_char_len_long  => char_len_long
use icepack_kinds, only: icepack_log_kind  => log_kind
use icepack_kinds, only: icepack_int_kind  => int_kind
use icepack_kinds, only: icepack_int8_kind => int8_kind
use icepack_kinds, only: icepack_real_kind => real_kind
use icepack_kinds, only: icepack_dbl_kind  => dbl_kind
use icepack_kinds, only: icepack_r16_kind  => r16_kind
```

icepack_tracers defines a handful of parameters that provide information about maximum array sizes for static dimensioning:

```
use icepack_tracers,   only: icepack_max_nbtrcr => max_nbtrcr
use icepack_tracers,   only: icepack_max_algae  => max_algae
use icepack_tracers,   only: icepack_max_dic    => max_dic
use icepack_tracers,   only: icepack_max_doc    => max_doc
use icepack_tracers,   only: icepack_max_don    => max_don
use icepack_tracers,   only: icepack_max_fe     => max_fe
use icepack_tracers,   only: icepack_max_aero   => max_aero
use icepack_tracers,   only: icepack_max_iso    => max_iso
use icepack_tracers,   only: icepack_nmodal1    => nmodal1
use icepack_tracers,   only: icepack_nmodal2    => nmodal2
use icepack_parameters,only: icepack_nspint     => nspint
```

icepack_parameters provides init, query, write, and recompute methods to define constant values and model parameters. These constants have defaults that the caller can query or reset:

```
use icepack_parameters, only: icepack_init_parameters
use icepack_parameters, only: icepack_query_parameters
use icepack_parameters, only: icepack_write_parameters
use icepack_parameters, only: icepack_recompute_constants
```

icepack_parameters also provides a set of constants:

```
use icepack_parameters, only: c0, c1, c1p5, c2, c3, c4, c5, c6, c8
use icepack_parameters, only: c10, c15, c16, c20, c25, c100, c1000
use icepack_parameters, only: p001, p01, p1, p2, p4, p5, p6, p05
use icepack_parameters, only: p15, p25, p75, p333, p666
```

icepack_tracers provides init, query, and write methods to define various tracer sizes, flags, and indices. The tracers have some defaults that the caller can query or reset:

```
use icepack_tracers, only: icepack_compute_tracers
use icepack_tracers, only: icepack_init_tracer_flags
use icepack_tracers, only: icepack_query_tracer_flags
use icepack_tracers, only: icepack_write_tracer_flags
use icepack_tracers, only: icepack_init_tracer_indices
use icepack_tracers, only: icepack_query_tracer_indices
use icepack_tracers, only: icepack_write_tracer_indices
use icepack_tracers, only: icepack_init_tracer_sizes
use icepack_tracers, only: icepack_query_tracer_sizes
use icepack_tracers, only: icepack_write_tracer_sizes
```

icepack_itd provides three public interfaces to compute the ice thickness distribution:

```
use icepack_itd, only: icepack_init_itd
use icepack_itd, only: icepack_init_itd_hist
use icepack_itd, only: icepack_aggregate
```

icepack_fsd provides three public interfaces to compute the floe size distribution:

```
use icepack_fsd, only: icepack_init_fsd_bounds
use icepack_fsd, only: icepack_init_fsd
use icepack_fsd, only: icepack_cleanup_fsd
```

icepack_mechred contains two public interfaces to compute ridging and ice strength:

```
use icepack_mechred, only: icepack_step_ridge
use icepack_mechred, only: icepack_ice_strength
```

icepack_wavefracspec provides two public interface to compute the impact of waves on sea ice:

```
use icepack_wavefracspec, only: icepack_init_wave
use icepack_wavefracspec, only: icepack_step_wavefracture
```

icepack_snow provides a routine to initialize the snow physics and a routine to update the snow physics:

```
use icepack_snow, only: icepack_init_snow
use icepack_snow, only: icepack_step_snow
```

icepack_shortwave provides a routine to initialize the radiation computation and a routine to update the radiation computation:

```
use icepack_shortwave, only: icepack_prep_radiation
use icepack_shortwave, only: icepack_step_radiation
```

icepack_brine addresses brine computations:

```
use icepack_brine, only: icepack_init_hbrine
use icepack_brine, only: icepack_init_zsalinity  ! DEPRECATED
```

icepack_zbgc contains several public interfaces to support initialization and computation for the skeletal layer bgc and zbgc options:

```
use icepack_zbgc , only: icepack_init_bgc
use icepack_zbgc , only: icepack_init_zbgc
use icepack_zbgc , only: icepack_biogeochemistry
use icepack_zbgc , only: icepack_init_ocean_bio
use icepack_zbgc , only: icepack_load_ocean_bio_array
```

There are a couple of routines to support computation of an atmosphere and ocean interaction:

```
use icepack_atmo , only: icepack_atm_boundary
use icepack_ocean, only: icepack_ocn_mixed_layer
```

icepack_orbital provides methods to set and query orbital parameters:

```
use icepack_orbital       , only: icepack_init_orbit
use icepack_orbital       , only: icepack_query_orbit
```

icepack_step_therm1 and icepack_step_therm2 compute the ice thermodynamics in two steps:

```
use icepack_therm_vertical, only: icepack_step_therm1
use icepack_therm_itd     , only: icepack_step_therm2
```

icepack_therm_shared provides several methods to compute different internal terms:

```
use icepack_therm_shared  , only: icepack_ice_temperature
use icepack_therm_shared  , only: icepack_snow_temperature
use icepack_therm_shared  , only: icepack_liquidus_temperature
use icepack_therm_shared  , only: icepack_sea_freezing_temperature
```

(continues on next page)

(continued from previous page)

```
use icepack_therm_shared  , only: icepack_enthalpy_snow
use icepack_therm_shared  , only: icepack_init_thermo
use icepack_therm_shared  , only: icepack_init_trcr
```

icepack_mushy_physics provides three public interfaces to compute various functions:

```
use icepack_mushy_physics , only: icepack_mushy_density_brine
use icepack_mushy_physics , only: icepack_mushy_liquid_fraction
use icepack_mushy_physics , only: icepack_mushy_temperature_mush
```

icepack_warnings provides several methods for getting, writing, and clearing messages. There is also a function that returns a logical flag indicating whether the column physics has aborted:

```
use icepack_warnings, only: icepack_warnings_clear
use icepack_warnings, only: icepack_warnings_print
use icepack_warnings, only: icepack_warnings_flush
use icepack_warnings, only: icepack_warnings_aborted
```

icepack_configure is a standalone icepack method that should always be called first:

```
public :: icepack_configure
```

The actual interfaces are documented in *Public Interfaces*

4.3.3 Calling Sequence

The calling sequence required to setup and run the column physics is generally described below. Several steps may be needed to be taken by the host between icepack calls in order to support the icepack interfaces. The icepack driver and the CICE model provide working examples of how to do this in practice. The sample below does not include bgc:

```
start driver

  call *icepack_configure*

initialize driver and read in driver namelist

  call *icepack_init_parameters*
  call *icepack_init_tracers_*
  call *icepack_init_trcr*
  call *icepack_init_thermo*
  call *icepack_init_itd*
  call *icepack_init_itd_hist*
  loop over gridcells
    call *icepack_step_radiation*
  end loop over gridcells
  call *icepack_init_hbrine*
  loop over gridcells
     call *icepack_aggregate*
  end loop over gridcells

  loop over timesteps
    loop over gridcells
```

(continues on next page)

(continued from previous page)

```
      call *icepack_prep_radiation*
      call *icepack_step_therm1*
      call *icepack_step_therm2*
      call *icepack_aggregate*
      call *icepack_step_ridge*
      call *icepack_step_radiation*
      call *icepack_atm_boundary*
      call *icepack_ocn_mixed_layer*
    end loop over gridcells
  end loop over timesteps

end driver
```

4.4 Public Interfaces

Below are a list of public icepack interfaces.

The documentation for these interfaces is extracted directly from the icepack source code using the script `doc/generate_interfaces.sh`. That script updates the rst file `interfaces.include` in the `doc/source/user_guide directory`. That file is part of the internal documentation. There is information about how `generate_interfaces.sh` parses the source code in a comment section in that script. Executing `icepack.setup --docintfc` will run the generate_interfaces script as noted in *Command Line Options*. Once `generate_interfaces` is executed, the user still has to git add, commit, and push the changes to the documentation manually. A typical workflow would be:

```
# verify all public interfaces in the columnphysics have appropriate autodocument␣
↪comment line
#  there should be a "!autodocument_start ${interface_name}" at the begining of the␣
↪interface
#  there should be a "!autodocument_end" at the end of the declaration of the interface␣
↪arguments
./icepack.setup --docintfc
git add doc/source/user_guide/interfaces.include
git commit -m "update public interface documentation"
```

4.4.1 icepack_atmo.F90

icepack_atm_boundary

```
!

      subroutine icepack_atm_boundary(sfctype,                   &
                                      Tsf,         potT,         &
                                      uatm,        vatm,         &
                                      wind,        zlvl,         &
                                      Qa,          rhoa,         &
                                      strx,        stry,         &
                                      Tref,        Qref,         &
                                      delt,        delq,         &
                                      lhcoef,      shcoef,       &
```

(continues on next page)

(continued from previous page)

```
                                      Cdn_atm,                    &
                                      Cdn_atm_ratio_n,            &
                                      Qa_iso,      Qref_iso,      &
                                      uvel,        vvel,          &
                                      Uref,        zlvs)

      character (len=3), intent(in) :: &
         sfctype      ! ice or ocean

      real (kind=dbl_kind), intent(in) :: &
         Tsf       , & ! surface temperature of ice or ocean
         potT      , & ! air potential temperature  (K)
         uatm      , & ! x-direction wind speed (m/s)
         vatm      , & ! y-direction wind speed (m/s)
         wind      , & ! wind speed (m/s)
         zlvl      , & ! atm level height for momentum (and scalars if zlvs is not␣
↪present) (m)
         Qa        , & ! specific humidity (kg/kg)
         rhoa          ! air density (kg/m^3)

      real (kind=dbl_kind), intent(inout) :: &
         Cdn_atm   , &    ! neutral drag coefficient
         Cdn_atm_ratio_n ! ratio drag coeff / neutral drag coeff

      real (kind=dbl_kind), intent(inout) :: &
         strx      , & ! x surface stress (N)
         stry          ! y surface stress (N)

      real (kind=dbl_kind), intent(inout) :: &
         Tref      , & ! reference height temperature  (K)
         Qref      , & ! reference height specific humidity (kg/kg)
         delt      , & ! potential T difference   (K)
         delq      , & ! humidity difference      (kg/kg)
         shcoef    , & ! transfer coefficient for sensible heat
         lhcoef        ! transfer coefficient for latent heat

      real (kind=dbl_kind), intent(in), dimension(:), optional :: &
         Qa_iso        ! specific isotopic humidity (kg/kg)

      real (kind=dbl_kind), intent(inout), dimension(:), optional :: &
         Qref_iso      ! reference specific isotopic humidity (kg/kg)

      real (kind=dbl_kind), intent(in), optional :: &
         uvel      , & ! x-direction ice speed (m/s)
         vvel      , & ! y-direction ice speed (m/s)
         zlvs          ! atm level height for scalars (if different than zlvl) (m)

      real (kind=dbl_kind), intent(out), optional :: &
         Uref          ! reference height wind speed (m/s)
```

4.4.2 icepack_brine.F90

icepack_init_hbrine

```
!  Initialize brine height tracer

      subroutine icepack_init_hbrine(bgrid, igrid, cgrid, &
          icgrid, swgrid, nblyr, nilyr, phi_snow)

      integer (kind=int_kind), intent(in) :: &
         nilyr, & ! number of ice layers
         nblyr    ! number of bio layers

      real (kind=dbl_kind), intent(inout) :: &
         phi_snow           ! porosity at the ice-snow interface

      real (kind=dbl_kind), dimension (nblyr+2), intent(out) :: &
         bgrid              ! biology nondimensional vertical grid points

      real (kind=dbl_kind), dimension (nblyr+1), intent(out) :: &
         igrid              ! biology vertical interface points

      real (kind=dbl_kind), dimension (nilyr+1), intent(out) :: &
         cgrid         , &  ! CICE vertical coordinate
         icgrid        , &  ! interface grid for CICE (shortwave variable)
         swgrid             ! grid for ice tracers used in dEdd scheme
```

icepack_init_zsalinity

```
!  **DEPRECATED**, all code removed
!  Interface provided for backwards compatibility

      subroutine icepack_init_zsalinity(nblyr,ntrcr_o,  Rayleigh_criteria, &
               Rayleigh_real, trcrn_bgc, nt_bgc_S, ncat, sss)

      integer (kind=int_kind), intent(in) :: &
       nblyr   , & ! number of biolayers
       ntrcr_o, & ! number of non bio tracers
       ncat    , & ! number of categories
       nt_bgc_S    ! zsalinity index

      logical (kind=log_kind), intent(inout) :: &
       Rayleigh_criteria

      real (kind=dbl_kind), intent(inout):: &
       Rayleigh_real

      real (kind=dbl_kind), intent(in):: &
       sss

      real (kind=dbl_kind), dimension(:,:), intent(inout):: &
       trcrn_bgc   ! bgc subset of trcrn
```

4.4.3 icepack_fsd.F90

icepack_init_fsd_bounds

```
!  Initialize ice fsd bounds (call whether or not restarting)
!  Define the bounds, midpoints and widths of floe size
!  categories in area and radius
!
!  authors: Lettie Roach, NIWA/VUW and C. M. Bitz, UW

      subroutine icepack_init_fsd_bounds(nfsd, &
         floe_rad_l,    &  ! fsd size lower bound in m (radius)
         floe_rad_c,    &  ! fsd size bin centre in m (radius)
         floe_binwidth, &  ! fsd size bin width in m (radius)
         c_fsd_range,   &  ! string for history output
         write_diags    )  ! flag for writing diagnostics

      integer (kind=int_kind), intent(in) :: &
         nfsd              ! number of floe size categories

      real(kind=dbl_kind), dimension(:), intent(inout) ::  &
         floe_rad_l,    &  ! fsd size lower bound in m (radius)
         floe_rad_c,    &  ! fsd size bin centre in m (radius)
         floe_binwidth     ! fsd size bin width in m (radius)

      character (len=35), intent(out) :: &
         c_fsd_range(nfsd) ! string for history output

      logical (kind=log_kind), intent(in), optional :: &
         write_diags       ! write diags flag
```

icepack_init_fsd

```
!
!  Initialize the FSD
!
!  authors: Lettie Roach, NIWA/VUW

      subroutine icepack_init_fsd(nfsd, ice_ic, &
         floe_rad_c,    &  ! fsd size bin centre in m (radius)
         floe_binwidth, &  ! fsd size bin width in m (radius)
         afsd)             ! floe size distribution tracer

      integer(kind=int_kind), intent(in) :: &
         nfsd

      character(len=char_len_long), intent(in) :: &
         ice_ic            ! method of ice cover initialization

      real(kind=dbl_kind), dimension(:), intent(inout) ::  &
         floe_rad_c,    &  ! fsd size bin centre in m (radius)
```

(continues on next page)

(continued from previous page)

```
         floe_binwidth     ! fsd size bin width in m (radius)

      real (kind=dbl_kind), dimension (:), intent(inout) :: &
         afsd              ! floe size tracer: fraction distribution of floes
```

icepack_cleanup_fsd

```
!
!  Clean up small values and renormalize
!
!  authors:  Elizabeth Hunke, LANL
!
      subroutine icepack_cleanup_fsd (ncat, nfsd, afsdn)

      integer (kind=int_kind), intent(in) :: &
         ncat           , & ! number of thickness categories
         nfsd               ! number of floe size categories

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         afsdn              ! floe size distribution tracer
```

4.4.4 icepack_intfc.F90

icepack_intfc.F90

```
! public parameters and interface routines for the icepack columnpackage code

      module icepack_intfc

      use icepack_kinds, only: icepack_char_len  => char_len
      use icepack_kinds, only: icepack_char_len_long  => char_len_long
      use icepack_kinds, only: icepack_log_kind  => log_kind
      use icepack_kinds, only: icepack_int_kind  => int_kind
      use icepack_kinds, only: icepack_int8_kind => int8_kind
      use icepack_kinds, only: icepack_real_kind => real_kind
      use icepack_kinds, only: icepack_dbl_kind  => dbl_kind
      use icepack_kinds, only: icepack_r16_kind  => r16_kind

      use icepack_tracers,    only: icepack_max_nbtrcr => max_nbtrcr
      use icepack_tracers,    only: icepack_max_algae  => max_algae
      use icepack_tracers,    only: icepack_max_dic    => max_dic
      use icepack_tracers,    only: icepack_max_doc    => max_doc
      use icepack_tracers,    only: icepack_max_don    => max_don
      use icepack_tracers,    only: icepack_max_fe     => max_fe
      use icepack_tracers,    only: icepack_max_aero   => max_aero
      use icepack_tracers,    only: icepack_max_iso    => max_iso
      use icepack_tracers,    only: icepack_nmodal1    => nmodal1
      use icepack_tracers,    only: icepack_nmodal2    => nmodal2
```

(continues on next page)

(continued from previous page)

```
      use icepack_shortwave_data, only: icepack_nspint_3bd => nspint_3bd
      use icepack_shortwave_data, only: icepack_nspint_5bd => nspint_5bd

      use icepack_parameters, only: icepack_init_parameters
      use icepack_parameters, only: icepack_query_parameters
      use icepack_parameters, only: icepack_write_parameters
      use icepack_parameters, only: icepack_recompute_constants
      use icepack_parameters, only: secday, spval_const
      use icepack_parameters, only: c0, c1, c1p5, c2, c3, c4, c5, c6, c8
      use icepack_parameters, only: c10, c15, c16, c20, c25, c100, c1000
      use icepack_parameters, only: p001, p01, p1, p2, p4, p5, p6, p05
      use icepack_parameters, only: p15, p25, p75, p333, p666

      use icepack_tracers, only: icepack_compute_tracers
      use icepack_tracers, only: icepack_init_tracer_flags
      use icepack_tracers, only: icepack_query_tracer_flags
      use icepack_tracers, only: icepack_write_tracer_flags
      use icepack_tracers, only: icepack_init_tracer_indices
      use icepack_tracers, only: icepack_query_tracer_indices
      use icepack_tracers, only: icepack_write_tracer_indices
      use icepack_tracers, only: icepack_init_tracer_sizes
      use icepack_tracers, only: icepack_query_tracer_sizes
      use icepack_tracers, only: icepack_write_tracer_sizes

      use icepack_itd, only: icepack_init_itd
      use icepack_itd, only: icepack_init_itd_hist
      use icepack_itd, only: icepack_aggregate

      use icepack_fsd, only: icepack_init_fsd_bounds
      use icepack_fsd, only: icepack_init_fsd
      use icepack_fsd, only: icepack_cleanup_fsd

      use icepack_mechred, only: icepack_step_ridge
      use icepack_mechred, only: icepack_ice_strength

      use icepack_wavefracspec, only: icepack_init_wave
      use icepack_wavefracspec, only: icepack_step_wavefracture

      use icepack_snow, only: icepack_init_snow
      use icepack_snow, only: icepack_step_snow

      use icepack_shortwave, only: icepack_init_radiation
      use icepack_shortwave, only: icepack_prep_radiation
      use icepack_shortwave, only: icepack_step_radiation

      use icepack_brine, only: icepack_init_hbrine
      use icepack_brine, only: icepack_init_zsalinity    ! deprecated

      use icepack_zbgc , only: icepack_init_bgc
      use icepack_zbgc , only: icepack_init_zbgc
      use icepack_zbgc , only: icepack_biogeochemistry
      use icepack_zbgc , only: icepack_init_ocean_bio
```

(continues on next page)

(continued from previous page)

```
      use icepack_zbgc , only: icepack_load_ocean_bio_array

      use icepack_atmo , only: icepack_atm_boundary
      use icepack_ocean, only: icepack_ocn_mixed_layer

      use icepack_orbital       , only: icepack_init_orbit
      use icepack_orbital       , only: icepack_query_orbit

      use icepack_therm_vertical, only: icepack_step_therm1
      use icepack_therm_itd     , only: icepack_step_therm2
      use icepack_therm_shared  , only: icepack_ice_temperature
      use icepack_therm_shared  , only: icepack_snow_temperature
      use icepack_therm_shared  , only: icepack_liquidus_temperature
      use icepack_therm_shared  , only: icepack_sea_freezing_temperature
      use icepack_therm_shared  , only: icepack_init_thermo
      use icepack_therm_shared  , only: icepack_salinity_profile
      use icepack_therm_shared  , only: icepack_init_trcr

      use icepack_mushy_physics , only: icepack_enthalpy_snow
      use icepack_mushy_physics , only: icepack_enthalpy_mush
      use icepack_mushy_physics , only: icepack_mushy_density_brine
      use icepack_mushy_physics , only: icepack_mushy_liquid_fraction
      use icepack_mushy_physics , only: icepack_mushy_temperature_mush

      use icepack_warnings, only: icepack_warnings_clear
      use icepack_warnings, only: icepack_warnings_print
      use icepack_warnings, only: icepack_warnings_flush
      use icepack_warnings, only: icepack_warnings_aborted
      use icepack_warnings, only: icepack_warnings_getall
```

4.4.5 icepack_itd.F90

icepack_init_itd

```
! Initialize area fraction and thickness boundaries for the itd model
!
! authors: William H. Lipscomb and Elizabeth C. Hunke, LANL
!          C. M. Bitz, UW

      subroutine icepack_init_itd(ncat, hin_max)

      integer (kind=int_kind), intent(in) :: &
           ncat ! number of thickness categories

      real (kind=dbl_kind), intent(out) :: &
           hin_max(0:ncat)  ! category limits (m)
```

icepack_init_itd_hist

```
! Initialize area fraction and thickness boundaries for the itd model
!
! authors: William H. Lipscomb and Elizabeth C. Hunke, LANL
!          C. M. Bitz, UW

      subroutine icepack_init_itd_hist (ncat, hin_max, c_hi_range)

      integer (kind=int_kind), intent(in) :: &
           ncat ! number of thickness categories

      real (kind=dbl_kind), intent(in) :: &
           hin_max(0:ncat)  ! category limits (m)

      character (len=35), intent(out) :: &
           c_hi_range(ncat) ! string for history output
```

icepack_aggregate

```
! Aggregate ice state variables over thickness categories.
!
! authors: C. M. Bitz, UW
!          W. H. Lipscomb, LANL

      subroutine icepack_aggregate (ncat,               &
                                    aicen,    trcrn,    &
                                    vicen,    vsnon,    &
                                    aice,     trcr,     &
                                    vice,     vsno,     &
                                    aice0,              &
                                    ntrcr,              &
                                    trcr_depend,        &
                                    trcr_base,          &
                                    n_trcr_strata,      &
                                    nt_strata, Tf)

      integer (kind=int_kind), intent(in) :: &
         ncat  , & ! number of thickness categories
         ntrcr     ! number of tracers in use

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         aicen , & ! concentration of ice
         vicen , & ! volume per unit area of ice          (m)
         vsnon     ! volume per unit area of snow         (m)

      real (kind=dbl_kind), dimension (:,:), intent(inout) :: &
         trcrn     ! ice tracers

      integer (kind=int_kind), dimension (:), intent(in) :: &
         trcr_depend, & ! = 0 for aicen tracers, 1 for vicen, 2 for vsnon
```

(continues on next page)

(continued from previous page)

```fortran
         n_trcr_strata  ! number of underlying tracer layers

      real (kind=dbl_kind), dimension (:,:), intent(in) :: &
         trcr_base      ! = 0 or 1 depending on tracer dependency
                        ! argument 2:  (1) aice, (2) vice, (3) vsno

      integer (kind=int_kind), dimension (:,:), intent(in) :: &
         nt_strata      ! indices of underlying tracer layers

      real (kind=dbl_kind), intent(out) :: &
         aice   , & ! concentration of ice
         vice   , & ! volume per unit area of ice          (m)
         vsno   , & ! volume per unit area of snow         (m)
         aice0      ! concentration of open water

      real (kind=dbl_kind), dimension (:), intent(out) :: &
         trcr      ! ice tracers

      real (kind=dbl_kind), intent(in) :: &
         Tf                ! freezing temperature
```

4.4.6 icepack_mechred.F90

icepack_ice_strength

```fortran
! Compute the strength of the ice pack, defined as the energy (J m-2)
! dissipated per unit area removed from the ice pack under compression,
! and assumed proportional to the change in potential energy caused
! by ridging.
!
! See Rothrock (1975) and Hibler (1980).
!
! For simpler strength parameterization, see this reference:
! Hibler, W. D. III, 1979: A dynamic-thermodynamic sea ice model,
!  J. Phys. Oceanog., 9, 817-846.
!
! authors: William H. Lipscomb, LANL
!          Elizabeth C. Hunke, LANL

      subroutine icepack_ice_strength (ncat,              &
                                       aice,     vice,     &
                                       aice0,    aicen,    &
                                       vicen,    &
                                       strength)

      integer (kind=int_kind), intent(in) :: &
         ncat       ! number of thickness categories

      real (kind=dbl_kind), intent(in) :: &
         aice    , & ! concentration of ice
```

(continues on next page)

(continued from previous page)

```
         vice   , & ! volume per unit area of ice  (m)
         aice0      ! concentration of open water

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         aicen  , & ! concentration of ice
         vicen      ! volume per unit area of ice  (m)

      real (kind=dbl_kind), intent(inout) :: &
         strength   ! ice strength (N/m)
```

icepack_step_ridge

```
! Computes sea ice mechanical deformation
!
! authors: William H. Lipscomb, LANL
!          Elizabeth C. Hunke, LANL

      subroutine icepack_step_ridge (dt,           ndtd,         &
                                     nilyr,        nslyr,        &
                                     nblyr,                      &
                                     ncat,         hin_max,      &
                                     rdg_conv,     rdg_shear,    &
                                     aicen,                      &
                                     trcrn,                      &
                                     vicen,        vsnon,        &
                                     aice0,        trcr_depend,  &
                                     trcr_base,    n_trcr_strata, &
                                     nt_strata,                  &
                                     dardg1dt,     dardg2dt,     &
                                     dvirdgdt,     opening,      &
                                     fpond,                      &
                                     fresh,        fhocn,        &
                                     n_aero,                     &
                                     faero_ocn,    fiso_ocn,     &
                                     aparticn,     krdgn,        &
                                     aredistn,     vredistn,     &
                                     dardg1ndt,    dardg2ndt,    &
                                     dvirdgndt,                  &
                                     araftn,       vraftn,       &
                                     aice,         fsalt,        &
                                     first_ice,    fzsal,        &
                                     flux_bio,     closing, Tf )

      real (kind=dbl_kind), intent(in) :: &
         dt           ! time step

      real (kind=dbl_kind), intent(in) :: &
         Tf           ! freezing temperature

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

(continues on next page)

(continued from previous page)

```
         ncat   , & ! number of thickness categories
         ndtd   , & ! number of dynamics supercycles
         nblyr  , & ! number of bio layers
         nilyr  , & ! number of ice layers
         nslyr  , & ! number of snow layers
         n_aero     ! number of aerosol tracers

      real (kind=dbl_kind), dimension(0:ncat), intent(inout) :: &
         hin_max    ! category limits (m)

      integer (kind=int_kind), dimension (:), intent(in) :: &
         trcr_depend, & ! = 0 for aicen tracers, 1 for vicen, 2 for vsnon
         n_trcr_strata  ! number of underlying tracer layers

      real (kind=dbl_kind), dimension (:,:), intent(in) :: &
         trcr_base      ! = 0 or 1 depending on tracer dependency
                        ! argument 2:  (1) aice, (2) vice, (3) vsno

      integer (kind=int_kind), dimension (:,:), intent(in) :: &
         nt_strata      ! indices of underlying tracer layers

      real (kind=dbl_kind), intent(inout) :: &
         aice      , & ! sea ice concentration
         aice0     , & ! concentration of open water
         rdg_conv  , & ! convergence term for ridging (1/s)
         rdg_shear, & ! shear term for ridging (1/s)
         dardg1dt  , & ! rate of area loss by ridging ice (1/s)
         dardg2dt  , & ! rate of area gain by new ridges (1/s)
         dvirdgdt  , & ! rate of ice volume ridged (m/s)
         opening   , & ! rate of opening due to divergence/shear (1/s)
         fpond     , & ! fresh water flux to ponds (kg/m^2/s)
         fresh     , & ! fresh water flux to ocean (kg/m^2/s)
         fsalt     , & ! salt flux to ocean (kg/m^2/s)
         fhocn         ! net heat flux to ocean (W/m^2)

      real (kind=dbl_kind), intent(inout), optional :: &
         fzsal         ! zsalinity flux to ocean(kg/m^2/s) (deprecated)

      real (kind=dbl_kind), intent(inout), optional :: &
         closing       ! rate of closing due to divergence/shear (1/s)

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         aicen     , & ! concentration of ice
         vicen     , & ! volume per unit area of ice          (m)
         vsnon     , & ! volume per unit area of snow         (m)
         dardg1ndt, & ! rate of area loss by ridging ice (1/s)
         dardg2ndt, & ! rate of area gain by new ridges (1/s)
         dvirdgndt, & ! rate of ice volume ridged (m/s)
         aparticn  , & ! participation function
         krdgn     , & ! mean ridge thickness/thickness of ridging ice
         araftn    , & ! rafting ice area
         vraftn    , & ! rafting ice volume
```

(continues on next page)

(continued from previous page)

```
         aredistn , & ! redistribution function: fraction of new ridge area
         vredistn , & ! redistribution function: fraction of new ridge volume
         faero_ocn, & ! aerosol flux to ocean  (kg/m^2/s)
         flux_bio     ! all bio fluxes to ocean

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         fiso_ocn     ! isotope flux to ocean  (kg/m^2/s)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         trcrn        ! tracers

      !logical (kind=log_kind), intent(in) :: &
         !tr_pond_topo,& ! if .true., use explicit topography-based ponds
         !tr_aero     ,& ! if .true., use aerosol tracers
         !tr_brine    !,& ! if .true., brine height differs from ice thickness

      logical (kind=log_kind), dimension(:), intent(inout) :: &
         first_ice    ! true until ice forms
```

4.4.7 icepack_mushy_physics.F90

icepack_mushy_density_brine

```
! Compute density of brine from brine salinity

  function icepack_mushy_density_brine(Sbr) result(rho)

    real(kind=dbl_kind), intent(in) :: &
         Sbr ! brine salinity (ppt)

    real(kind=dbl_kind) :: &
         rho ! brine density (kg m-3)
```

icepack_enthalpy_snow

```
! Enthalpy of snow from snow temperature

  function icepack_enthalpy_snow(zTsn) result(zqsn)

    real(kind=dbl_kind), intent(in) :: &
         zTsn ! snow layer temperature (C)

    real(kind=dbl_kind) :: &
         zqsn ! snow layer enthalpy (J m-3)
```

icepack_enthalpy_mush

```
! Enthalpy of mush from mush temperature and bulk salinity

  function icepack_enthalpy_mush(zTin, zSin) result(zqin)

    real(kind=dbl_kind), intent(in) :: &
         zTin, & ! ice layer temperature (C)
         zSin    ! ice layer bulk salinity (ppt)

    real(kind=dbl_kind) :: &
         zqin    ! ice layer enthalpy (J m-3)
```

icepack_mushy_temperature_mush

```
! Temperature of mush from mush enthalpy and bulk salinity

  function icepack_mushy_temperature_mush(zqin, zSin) result(zTin)

    real(kind=dbl_kind), intent(in) :: &
         zqin   , & ! ice enthalpy (J m-3)
         zSin       ! ice layer bulk salinity (ppt)

    real(kind=dbl_kind) :: &
         zTin       ! ice layer temperature (C)
```

icepack_mushy_liquid_fraction

```
! Liquid fraction of mush from mush temperature and bulk salinity

  function icepack_mushy_liquid_fraction(zTin, zSin) result(phi)

    real(kind=dbl_kind), intent(in) :: &
         zTin, & ! ice layer temperature (C)
         zSin    ! ice layer bulk salinity (ppt)

    real(kind=dbl_kind) :: &
         phi     ! liquid fraction
```

4.4.8 icepack_ocean.F90

icepack_ocn_mixed_layer

```
! Compute the mixed layer heat balance and update the SST.
! Compute the energy available to freeze or melt ice.
! NOTE: SST changes due to fluxes through the ice are computed in
!       icepack_therm_vertical.

```

(continues on next page)

(continued from previous page)

```
subroutine icepack_ocn_mixed_layer (alvdr_ocn, swvdr,      &
                                    alidr_ocn, swidr,      &
                                    alvdf_ocn, swvdf,      &
                                    alidf_ocn, swidf,      &
                                    sst,       flwout_ocn, &
                                    fsens_ocn, shcoef,     &
                                    flat_ocn,  lhcoef,     &
                                    evap_ocn,  flw,        &
                                    delt,      delq,       &
                                    aice,      fhocn,      &
                                    fswthru,   hmix,       &
                                    Tf,        qdp,        &
                                    frzmlt,    dt)

real (kind=dbl_kind), intent(in) :: &
   alvdr_ocn , & ! visible, direct   (fraction)
   alidr_ocn , & ! near-ir, direct   (fraction)
   alvdf_ocn , & ! visible, diffuse  (fraction)
   alidf_ocn , & ! near-ir, diffuse  (fraction)
   swvdr     , & ! sw down, visible, direct  (W/m^2)
   swvdf     , & ! sw down, visible, diffuse (W/m^2)
   swidr     , & ! sw down, near IR, direct  (W/m^2)
   swidf     , & ! sw down, near IR, diffuse (W/m^2)
   flw       , & ! incoming longwave radiation (W/m^2)
   Tf        , & ! freezing temperature (C)
   hmix      , & ! mixed layer depth (m)
   delt      , & ! potential temperature difference   (K)
   delq      , & ! specific humidity difference   (kg/kg)
   shcoef    , & ! transfer coefficient for sensible heat
   lhcoef    , & ! transfer coefficient for latent heat
   fhocn     , & ! net heat flux to ocean (W/m^2)
   fswthru   , & ! shortwave penetrating to ocean (W/m^2)
   aice      , & ! ice area fraction
   dt            ! time step (s)

real (kind=dbl_kind), intent(inout) :: &
   flwout_ocn, & ! outgoing longwave radiation (W/m^2)
   fsens_ocn , & ! sensible heat flux (W/m^2)
   flat_ocn  , & ! latent heat flux   (W/m^2)
   evap_ocn  , & ! evaporative water flux (kg/m^2/s)
   qdp       , & ! deep ocean heat flux (W/m^2), negative upward
   sst       , & ! sea surface temperature (C)
   frzmlt        ! freezing/melting potential (W/m^2)
```

4.4.9 icepack_orbital.F90

icepack_init_orbit

```
! Compute orbital parameters for the specified date.

      subroutine icepack_init_orbit(iyear_AD_in, eccen_in, obliqr_in, &
         lambm0_in, mvelpp_in, obliq_in, mvelp_in, decln_in, eccf_in, &
         log_print_in)

      integer(kind=int_kind), optional, intent(in) :: iyear_AD_in  ! Year to calculate␣
↪orbit for
      real(kind=dbl_kind), optional, intent(in) :: eccen_in  ! Earth's orbital␣
↪eccentricity
      real(kind=dbl_kind), optional, intent(in) :: obliqr_in ! Earth's obliquity in␣
↪radians
      real(kind=dbl_kind), optional, intent(in) :: lambm0_in ! Mean longitude of␣
↪perihelion at the
                                                             ! vernal equinox (radians)
      real(kind=dbl_kind), optional, intent(in) :: mvelpp_in ! Earth's moving vernal␣
↪equinox longitude
                                                             ! of perihelion + pi␣
↪(radians)
      real(kind=dbl_kind), optional, intent(in) :: obliq_in  ! obliquity in degrees
      real(kind=dbl_kind), optional, intent(in) :: mvelp_in  ! moving vernal equinox long
      real(kind=dbl_kind), optional, intent(in) :: decln_in  ! solar declination angle␣
↪in radians
      real(kind=dbl_kind), optional, intent(in) :: eccf_in   ! earth orbit eccentricity␣
↪factor
      logical(kind=log_kind), optional, intent(in) :: log_print_in ! Flags print of␣
↪status/error
```

icepack_query_orbit

```
! Compute orbital parameters for the specified date.

      subroutine icepack_query_orbit(iyear_AD_out, eccen_out, obliqr_out, &
         lambm0_out, mvelpp_out, obliq_out, mvelp_out, decln_out, eccf_out, &
         log_print_out)

      integer(kind=int_kind), optional, intent(out) :: iyear_AD_out  ! Year to calculate␣
↪orbit for
      real(kind=dbl_kind), optional, intent(out) :: eccen_out  ! Earth's orbital␣
↪eccentricity
      real(kind=dbl_kind), optional, intent(out) :: obliqr_out ! Earth's obliquity in␣
↪radians
      real(kind=dbl_kind), optional, intent(out) :: lambm0_out ! Mean longitude of␣
↪perihelion at the
                                                             ! vernal equinox (radians)
      real(kind=dbl_kind), optional, intent(out) :: mvelpp_out ! Earth's moving vernal␣
↪equinox longitude
```

(continues on next page)

(continued from previous page)

```
                                                          ! of perihelion + pi␣
↪(radians)
     real(kind=dbl_kind), optional, intent(out) :: obliq_out  ! obliquity in degrees
     real(kind=dbl_kind), optional, intent(out) :: mvelp_out  ! moving vernal equinox␣
↪long
     real(kind=dbl_kind), optional, intent(out) :: decln_out  ! solar declination angle␣
↪in radians
     real(kind=dbl_kind), optional, intent(out) :: eccf_out   ! earth orbit␣
↪eccentricity factor
     logical(kind=log_kind), optional, intent(out) :: log_print_out ! Flags print of␣
↪status/error
```

4.4.10 icepack_parameters.F90

icepack_init_parameters

```
! subroutine to set the column package internal parameters

     subroutine icepack_init_parameters(   &
        argcheck_in, puny_in, bignum_in, pi_in, secday_in, &
        rhos_in, rhoi_in, rhow_in, cp_air_in, emissivity_in, &
        cp_ice_in, cp_ocn_in, hfrazilmin_in, floediam_in, &
        depressT_in, dragio_in, thickness_ocn_layer1_in, iceruf_ocn_in, &
        albocn_in, gravit_in, viscosity_dyn_in, tscale_pnd_drain_in, &
        Tocnfrz_in, rhofresh_in, zvir_in, vonkar_in, cp_wv_in, &
        stefan_boltzmann_in, ice_ref_salinity_in, &
        Tffresh_in, Lsub_in, Lvap_in, Timelt_in, Tsmelt_in, &
        iceruf_in, Cf_in, Pstar_in, Cstar_in, kappav_in, &
        kice_in, ksno_in, &
        zref_in, hs_min_in, snowpatch_in, rhosi_in, sk_l_in, &
        saltmax_in, phi_init_in, min_salin_in, salt_loss_in, &
        Tliquidus_max_in, &
        min_bgc_in, dSin0_frazil_in, hi_ssl_in, hs_ssl_in, &
        awtvdr_in, awtidr_in, awtvdf_in, awtidf_in, &
        qqqice_in, TTTice_in, qqqocn_in, TTTocn_in, &
        ktherm_in, conduct_in, fbot_xfer_type_in, calc_Tsfc_in, dts_b_in, &
        update_ocn_f_in, ustar_min_in, hi_min_in, a_rapid_mode_in, &
        cpl_frazil_in, &
        Rac_rapid_mode_in, aspect_rapid_mode_in, &
        dSdt_slow_mode_in, phi_c_slow_mode_in, &
        phi_i_mushy_in, shortwave_in, albedo_type_in, albsnowi_in, &
        albicev_in, albicei_in, albsnowv_in, &
        ahmax_in, R_ice_in, R_pnd_in, R_snw_in, dT_mlt_in, rsnw_mlt_in, &
        kalg_in, kstrength_in, krdg_partic_in, krdg_redist_in, mu_rdg_in, &
        atmbndy_in, calc_strair_in, formdrag_in, highfreq_in, natmiter_in, &
        atmiter_conv_in, calc_dragio_in, &
        tfrz_option_in, kitd_in, kcatbound_in, hs0_in, frzpnd_in, &
        saltflux_option_in, &
        floeshape_in, wave_spec_in, wave_spec_type_in, nfreq_in, &
        dpscale_in, rfracmin_in, rfracmax_in, pndaspect_in, hs1_in, hp1_in, &
```

(continues on next page)

(continued from previous page)

```
         bgc_flux_type_in, z_tracers_in, scale_bgc_in, solve_zbgc_in, &
         modal_aero_in, skl_bgc_in, solve_zsal_in, grid_o_in, l_sk_in, &
         initbio_frac_in, grid_oS_in, l_skS_in,  dEdd_algae_in, &
         phi_snow_in, T_max_in, fsal_in, &
         fr_resp_in, algal_vel_in, R_dFe2dust_in, dustFe_sol_in, &
         op_dep_min_in, fr_graze_s_in, fr_graze_e_in, fr_mort2min_in, &
         fr_dFe_in, k_nitrif_in, t_iron_conv_in, max_loss_in, &
         max_dfe_doc1_in, fr_resp_s_in, conserv_check_in, &
         y_sk_DMS_in, t_sk_conv_in, t_sk_ox_in, frazil_scav_in, &
         sw_redist_in, sw_frac_in, sw_dtemp_in, snwgrain_in, &
         snwredist_in, use_smliq_pnd_in, rsnw_fall_in, rsnw_tmax_in, &
         rhosnew_in, rhosmin_in, rhosmax_in, windmin_in, drhosdwind_in, &
         snwlvlfac_in, isnw_T_in, isnw_Tgrd_in, isnw_rhos_in, &
         snowage_rhos_in, snowage_Tgrd_in, snowage_T_in, &
         snowage_tau_in, snowage_kappa_in, snowage_drdt0_in, &
         snw_aging_table_in, snw_ssp_table_in )

      !-----------------------------------------------------------------
      ! control settings
      !-----------------------------------------------------------------

      character(len=*), intent(in), optional :: &
         argcheck_in      ! optional argument checking, never, first, or always

      !-----------------------------------------------------------------
      ! parameter constants
      !-----------------------------------------------------------------

      real (kind=dbl_kind), intent(in), optional :: &
         secday_in,     & !
         puny_in,       & !
         bignum_in,     & !
         pi_in            !

      !-----------------------------------------------------------------
      ! densities
      !-----------------------------------------------------------------

      real (kind=dbl_kind), intent(in), optional :: &
         rhos_in,       & ! density of snow (kg/m^3)
         rhoi_in,       & ! density of ice (kg/m^3)
         rhosi_in,      & ! average sea ice density (kg/m2)
         rhow_in,       & ! density of seawater (kg/m^3)
         rhofresh_in      ! density of fresh water (kg/m^3)

!-----------------------------------------------------------------------
! Parameters for thermodynamics
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(in), optional :: &
         floediam_in,   & ! effective floe diameter for lateral melt (m)
         hfrazilmin_in, & ! min thickness of new frazil ice (m)
```

(continues on next page)

(continued from previous page)

```
      cp_ice_in,     & ! specific heat of fresh ice (J/kg/K)
      cp_ocn_in,     & ! specific heat of ocn    (J/kg/K)
      depressT_in,   & ! Tf:brine salinity ratio (C/ppt)
      viscosity_dyn_in, & ! dynamic viscosity of brine (kg/m/s)
      tscale_pnd_drain_in,&! mushy macroscopic drainage timescale (days)
      Tocnfrz_in,    & ! freezing temp of seawater (C)
      Tffresh_in,    & ! freezing temp of fresh ice (K)
      Lsub_in,       & ! latent heat, sublimation freshwater (J/kg)
      Lvap_in,       & ! latent heat, vaporization freshwater (J/kg)
      Timelt_in,     & ! melting temperature, ice top surface  (C)
      Tsmelt_in,     & ! melting temperature, snow top surface (C)
      ice_ref_salinity_in, & ! (ppt)
      kice_in,       & ! thermal conductivity of fresh ice(W/m/deg)
      ksno_in,       & ! thermal conductivity of snow  (W/m/deg)
      hs_min_in,     & ! min snow thickness for computing zTsn (m)
      snowpatch_in,  & ! parameter for fractional snow area (m)
      saltmax_in,    & ! max salinity at ice base for BL99 (ppt)
      phi_init_in,   & ! initial liquid fraction of frazil
      min_salin_in,  & ! threshold for brine pocket treatment
      salt_loss_in,  & ! fraction of salt retained in zsalinity
      Tliquidus_max_in, & ! maximum liquidus temperature of mush (C)
      dSin0_frazil_in  ! bulk salinity reduction of newly formed frazil

   integer (kind=int_kind), intent(in), optional :: &
      ktherm_in          ! type of thermodynamics
                         ! -1 none
                         ! 1 = Bitz and Lipscomb 1999
                         ! 2 = mushy layer theory

   character (len=*), intent(in), optional :: &
      conduct_in, &        ! 'MU71' or 'bubbly'
      fbot_xfer_type_in, & ! transfer coefficient type for ice-ocean heat flux
      cpl_frazil_in        ! type of coupling for frazil ice

   logical (kind=log_kind), intent(in), optional :: &
      calc_Tsfc_in    , &! if true, calculate surface temperature
                         ! if false, Tsfc is computed elsewhere and
                         ! atmos-ice fluxes are provided to CICE
      update_ocn_f_in    ! include fresh water and salt fluxes for frazil

   real (kind=dbl_kind), intent(in), optional :: &
      dts_b_in,   &      ! zsalinity timestep
      hi_min_in,  &      ! minimum ice thickness allowed (m) for thermo
      ustar_min_in       ! minimum friction velocity for ice-ocean heat flux

   ! mushy thermo
   real(kind=dbl_kind), intent(in), optional :: &
      a_rapid_mode_in      , & ! channel radius for rapid drainage mode (m)
      Rac_rapid_mode_in    , & ! critical Rayleigh number for rapid drainage mode
      aspect_rapid_mode_in , & ! aspect ratio for rapid drainage mode (larger=wider)
      dSdt_slow_mode_in    , & ! slow mode drainage strength (m s-1 K-1)
      phi_c_slow_mode_in   , & ! liquid fraction porosity cutoff for slow mode
```

(continues on next page)

(continued from previous page)

```
         phi_i_mushy_in           ! liquid fraction of congelation ice

      character(len=*), intent(in), optional :: &
         tfrz_option_in              ! form of ocean freezing temperature
                                     ! 'minus1p8' = -1.8 C
                                     ! 'linear_salt' = -depressT * sss
                                     ! 'mushy' conforms with ktherm=2

      character(len=*), intent(in), optional :: &
         saltflux_option_in         ! Salt flux computation
                                    ! 'constant' reference value of ice_ref_salinity
                                    ! 'prognostic' prognostic salt flux

!-----------------------------------------------------------------------
! Parameters for radiation
!-----------------------------------------------------------------------

      real(kind=dbl_kind), intent(in), optional :: &
         emissivity_in, & ! emissivity of snow and ice
         albocn_in,     & ! ocean albedo
         vonkar_in,     & ! von Karman constant
         stefan_boltzmann_in, & !  W/m^2/K^4
         kappav_in,     & ! vis extnctn coef in ice, wvlngth<700nm (1/m)
         hi_ssl_in,     & ! ice surface scattering layer thickness (m)
         hs_ssl_in,     & ! visible, direct
         awtvdr_in,     & ! visible, direct  ! for history and
         awtidr_in,     & ! near IR, direct  ! diagnostics
         awtvdf_in,     & ! visible, diffuse
         awtidf_in        ! near IR, diffuse

      character (len=*), intent(in), optional :: &
         shortwave_in, & ! shortwave method, 'ccsm3' or 'dEdd' or 'dEdd_snicar_ad'
         albedo_type_in  ! albedo parameterization, 'ccsm3' or 'constant'
                         ! shortwave='dEdd' overrides this parameter

      ! baseline albedos for ccsm3 shortwave, set in namelist
      real (kind=dbl_kind), intent(in), optional :: &
         albicev_in   , & ! visible ice albedo for h > ahmax
         albicei_in   , & ! near-ir ice albedo for h > ahmax
         albsnowv_in  , & ! cold snow albedo, visible
         albsnowi_in  , & ! cold snow albedo, near IR
         ahmax_in         ! thickness above which ice albedo is constant (m)

      ! dEdd tuning parameters, set in namelist
      real (kind=dbl_kind), intent(in), optional :: &
         R_ice_in     , & ! sea ice tuning parameter; +1 > 1sig increase in albedo
         R_pnd_in     , & ! ponded ice tuning parameter; +1 > 1sig increase in albedo
         R_snw_in     , & ! snow tuning parameter; +1 > ~.01 change in broadband albedo
         dT_mlt_in    , & ! change in temp for non-melt to melt snow grain
                          ! radius change (C)
         rsnw_mlt_in  , & ! maximum melting snow grain radius (10^-6 m)
         kalg_in          ! algae absorption coefficient for 0.5 m thick layer
```

(continues on next page)

(continued from previous page)

```
      logical (kind=log_kind), intent(in), optional :: &
         sw_redist_in    ! redistribute shortwave

      real (kind=dbl_kind), intent(in), optional :: &
         sw_frac_in   , & ! Fraction of internal shortwave moved to surface
         sw_dtemp_in     ! temperature difference from melting

!-----------------------------------------------------------------------
! Parameters for dynamics
!-----------------------------------------------------------------------

      real(kind=dbl_kind), intent(in), optional :: &
         Cf_in,          & ! ratio of ridging work to PE change in ridging
         Pstar_in,       & ! constant in Hibler strength formula
         Cstar_in,       & ! constant in Hibler strength formula
         dragio_in,      & ! ice-ocn drag coefficient
         thickness_ocn_layer1_in, & ! thickness of first ocean level (m)
         iceruf_ocn_in, & ! under-ice roughness (m)
         gravit_in,      & ! gravitational acceleration (m/s^2)
         iceruf_in         ! ice surface roughness (m)

      integer (kind=int_kind), intent(in), optional :: & ! defined in namelist
         kstrength_in   , & ! 0 for simple Hibler (1979) formulation
                            ! 1 for Rothrock (1975) pressure formulation
         krdg_partic_in, & ! 0 for Thorndike et al. (1975) formulation
                            ! 1 for exponential participation function
         krdg_redist_in     ! 0 for Hibler (1980) formulation
                            ! 1 for exponential redistribution function

      real (kind=dbl_kind), intent(in), optional :: &
         mu_rdg_in          ! gives e-folding scale of ridged ice (m^.5)
                            ! (krdg_redist = 1)

      logical (kind=log_kind), intent(in), optional :: &
         calc_dragio_in     ! if true, calculate dragio from iceruf_ocn and thickness_ocn_
↪layer1

!-----------------------------------------------------------------------
! Parameters for atmosphere
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(in), optional :: &
         cp_air_in,      & ! specific heat of air (J/kg/K)
         cp_wv_in,       & ! specific heat of water vapor (J/kg/K)
         zvir_in,        & ! rh2o/rair - 1.0
         zref_in,        & ! reference height for stability (m)
         qqqice_in,      & ! for qsat over ice
         TTTice_in,      & ! for qsat over ice
         qqqocn_in,      & ! for qsat over ocn
         TTTocn_in         ! for qsat over ocn
```

(continues on next page)

(continued from previous page)

```
      character (len=*), intent(in), optional :: &
         atmbndy_in    ! atmo boundary method, 'similarity', 'constant' or 'mixed'

      logical (kind=log_kind), intent(in), optional :: &
         calc_strair_in,     & ! if true, calculate wind stress components
         formdrag_in,        & ! if true, calculate form drag
         highfreq_in           ! if true, use high frequency coupling

      integer (kind=int_kind), intent(in), optional :: &
         natmiter_in        ! number of iterations for boundary layer calculations

      ! Flux convergence tolerance
      real (kind=dbl_kind), intent(in), optional :: atmiter_conv_in

!-----------------------------------------------------------------------
! Parameters for the ice thickness distribution
!-----------------------------------------------------------------------

      integer (kind=int_kind), intent(in), optional :: &
         kitd_in        , & ! type of itd conversions
                            !   0 = delta function
                            !   1 = linear remap
         kcatbound_in       !   0 = old category boundary formula
                            !   1 = new formula giving round numbers
                            !   2 = WMO standard
                            !   3 = asymptotic formula

!-----------------------------------------------------------------------
! Parameters for the floe size distribution
!-----------------------------------------------------------------------

      integer (kind=int_kind), intent(in), optional :: &
         nfreq_in           ! number of frequencies

      real (kind=dbl_kind), intent(in), optional :: &
         floeshape_in       ! constant from Steele (unitless)

      logical (kind=log_kind), intent(in), optional :: &
         wave_spec_in       ! if true, use wave forcing

      character (len=*), intent(in), optional :: &
         wave_spec_type_in  ! type of wave spectrum forcing

!-----------------------------------------------------------------------
! Parameters for biogeochemistry
!-----------------------------------------------------------------------

     character (len=*), intent(in), optional :: &
        bgc_flux_type_in    ! type of ocean-ice piston velocity
                            ! 'constant', 'Jin2006'

      logical (kind=log_kind), intent(in), optional :: &
```

(continues on next page)

(continued from previous page)

```
         z_tracers_in,      & ! if .true., bgc or aerosol tracers are vertically resolved
         scale_bgc_in,      & ! if .true., initialize bgc tracers proportionally with␣
↪salinity
         solve_zbgc_in,     & ! if .true., solve vertical biochemistry portion of code
         dEdd_algae_in,     & ! if .true., algal absorptionof Shortwave is computed in␣
↪the
         modal_aero_in,     & ! if .true., use modal aerosol formulation in shortwave
         conserv_check_in     ! if .true., run conservation checks and abort if checks␣
↪fail

      logical (kind=log_kind), intent(in), optional :: &
         skl_bgc_in,        &   ! if true, solve skeletal biochemistry
         solve_zsal_in          ! if true, update salinity profile from solve_S_dt

      real (kind=dbl_kind), intent(in), optional :: &
         grid_o_in      , & ! for bottom flux
         l_sk_in        , & ! characteristic diffusive scale (zsalinity) (m)
         initbio_frac_in, & ! fraction of ocean tracer concentration used to initialize␣
↪tracer
         phi_snow_in        ! snow porosity at the ice/snow interface

      real (kind=dbl_kind), intent(in), optional :: &
         grid_oS_in     , & ! for bottom flux (zsalinity)
         l_skS_in           ! 0.02 characteristic skeletal layer thickness (m)␣
↪(zsalinity)
      real (kind=dbl_kind), intent(in), optional :: &
         fr_resp_in           , &   ! fraction of algal growth lost due to respiration
         algal_vel_in         , &   ! 0.5 cm/d(m/s) Lavoie 2005  1.5 cm/day
         R_dFe2dust_in        , &   !  g/g (3.5% content) Tagliabue 2009
         dustFe_sol_in        , &   ! solubility fraction
         T_max_in            , & ! maximum temperature (C)
         fsal_in             , & ! Salinity limitation (ppt)
         op_dep_min_in       , & ! Light attenuates for optical depths exceeding min
         fr_graze_s_in       , & ! fraction of grazing spilled or slopped
         fr_graze_e_in       , & ! fraction of assimilation excreted
         fr_mort2min_in      , & ! fractionation of mortality to Am
         fr_dFe_in           , & ! fraction of remineralized nitrogen
                                   ! (in units of algal iron)
         k_nitrif_in         , & ! nitrification rate (1/day)
         t_iron_conv_in      , & ! desorption loss pFe to dFe (day)
         max_loss_in         , & ! restrict uptake to % of remaining value
         max_dfe_doc1_in     , & ! max ratio of dFe to saccharides in the ice
                                   ! (nM Fe/muM C)
         fr_resp_s_in        , & ! DMSPd fraction of respiration loss as DMSPd
         y_sk_DMS_in         , & ! fraction conversion given high yield
         t_sk_conv_in        , & ! Stefels conversion time (d)
         t_sk_ox_in          , & ! DMS oxidation time (d)
         frazil_scav_in          ! scavenging fraction or multiple in frazil ice

      real (kind=dbl_kind), intent(in), optional :: &
         sk_l_in,       & ! skeletal layer thickness (m)
         min_bgc_in       ! fraction of ocean bgc concentration in surface melt
```

(continues on next page)

(continued from previous page)

```
!-----------------------------------------------------------------------
! Parameters for melt ponds
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(in), optional :: &
         hs0_in             ! snow depth for transition to bare sea ice (m)

      ! level-ice ponds
      character (len=*), intent(in), optional :: &
         frzpnd_in          ! pond refreezing parameterization

      real (kind=dbl_kind), intent(in), optional :: &
         dpscale_in, &      ! alter e-folding time scale for flushing
         rfracmin_in, &     ! minimum retained fraction of meltwater
         rfracmax_in, &     ! maximum retained fraction of meltwater
         pndaspect_in, &    ! ratio of pond depth to pond fraction
         hs1_in             ! tapering parameter for snow on pond ice

      ! topo ponds
      real (kind=dbl_kind), intent(in), optional :: &
         hp1_in             ! critical parameter for pond ice thickness

!-----------------------------------------------------------------------
! Parameters for snow redistribution, metamorphosis
!-----------------------------------------------------------------------

      character (len=*), intent(in), optional :: &
         snwredist_in, &    ! type of snow redistribution
         snw_aging_table_in ! snow aging lookup table

      logical (kind=log_kind), intent(in), optional :: &
         use_smliq_pnd_in, &! use liquid in snow for ponds
         snwgrain_in        ! snow metamorphosis

      real (kind=dbl_kind), intent(in), optional :: &
         rsnw_fall_in, &    ! radius of new snow (10^-6 m)
         rsnw_tmax_in, &    ! maximum snow radius (10^-6 m)
         rhosnew_in, &      ! new snow density (kg/m^3)
         rhosmin_in, &      ! minimum snow density (kg/m^3)
         rhosmax_in, &      ! maximum snow density (kg/m^3)
         windmin_in, &      ! minimum wind speed to compact snow (m/s)
         drhosdwind_in, &   ! wind compaction factor (kg s/m^4)
         snwlvlfac_in       ! fractional increase in snow depth

      integer (kind=int_kind), intent(in), optional :: &
         isnw_T_in, &       ! maxiumum temperature index
         isnw_Tgrd_in, &    ! maxiumum temperature gradient index
         isnw_rhos_in       ! maxiumum snow density index

      real (kind=dbl_kind), dimension(:), intent(in), optional :: &
         snowage_rhos_in, & ! snowage dimension data
```

(continues on next page)

(continued from previous page)

```
         snowage_Tgrd_in, & !
         snowage_T_in       !

      real (kind=dbl_kind), dimension(:,:,:), intent(in), optional :: &
         snowage_tau_in, &  ! (10^-6 m)
         snowage_kappa_in, &!
         snowage_drdt0_in   ! (10^-6 m/hr)

      character (len=char_len), intent(in), optional :: &
         snw_ssp_table_in   ! lookup table: 'snicar' or 'test'
```

icepack_query_parameters

```
! subroutine to query the column package internal parameters

      subroutine icepack_query_parameters(   &
         argcheck_out, puny_out, bignum_out, pi_out, rad_to_deg_out,&
         secday_out, c0_out, c1_out, c1p5_out, c2_out, c3_out, c4_out, &
         c5_out, c6_out, c8_out, c10_out, c15_out, c16_out, c20_out, &
         c25_out, c100_out, c180_out, c1000_out, p001_out, p01_out, p1_out, &
         p2_out, p4_out, p5_out, p6_out, p05_out, p15_out, p25_out, p75_out, &
         p333_out, p666_out, spval_const_out, pih_out, piq_out, pi2_out, &
         rhos_out, rhoi_out, rhow_out, cp_air_out, emissivity_out, &
         cp_ice_out, cp_ocn_out, hfrazilmin_out, floediam_out, &
         depressT_out, dragio_out, thickness_ocn_layer1_out, iceruf_ocn_out, &
         albocn_out, gravit_out, viscosity_dyn_out, tscale_pnd_drain_out, &
         Tocnfrz_out, rhofresh_out, zvir_out, vonkar_out, cp_wv_out, &
         stefan_boltzmann_out, ice_ref_salinity_out, &
         Tffresh_out, Lsub_out, Lvap_out, Timelt_out, Tsmelt_out, &
         iceruf_out, Cf_out, Pstar_out, Cstar_out, kappav_out, &
         kice_out, ksno_out, &
         zref_out, hs_min_out, snowpatch_out, rhosi_out, sk_l_out, &
         saltmax_out, phi_init_out, min_salin_out, salt_loss_out, &
         Tliquidus_max_out, &
         min_bgc_out, dSin0_frazil_out, hi_ssl_out, hs_ssl_out, &
         awtvdr_out, awtidr_out, awtvdf_out, awtidf_out, cpl_frazil_out, &
         qqqice_out, TTTice_out, qqqocn_out, TTTocn_out, update_ocn_f_out, &
         Lfresh_out, cprho_out, Cp_out, ustar_min_out, hi_min_out, a_rapid_mode_out, &
         ktherm_out, conduct_out, fbot_xfer_type_out, calc_Tsfc_out, dts_b_out, &
         Rac_rapid_mode_out, aspect_rapid_mode_out, dSdt_slow_mode_out, &
         phi_c_slow_mode_out, phi_i_mushy_out, shortwave_out, &
         albedo_type_out, albicev_out, albicei_out, albsnowv_out, &
         albsnowi_out, ahmax_out, R_ice_out, R_pnd_out, R_snw_out, dT_mlt_out, &
         rsnw_mlt_out, dEdd_algae_out, &
         kalg_out, kstrength_out, krdg_partic_out, krdg_redist_out, mu_rdg_out, &
         atmbndy_out, calc_strair_out, formdrag_out, highfreq_out, natmiter_out, &
         atmiter_conv_out, calc_dragio_out, &
         tfrz_option_out, kitd_out, kcatbound_out, hs0_out, frzpnd_out, &
         saltflux_option_out, &
         floeshape_out, wave_spec_out, wave_spec_type_out, nfreq_out, &
```

(continues on next page)

(continued from previous page)

```
   dpscale_out, rfracmin_out, rfracmax_out, pndaspect_out, hs1_out, hp1_out, &
   bgc_flux_type_out, z_tracers_out, scale_bgc_out, solve_zbgc_out, &
   modal_aero_out, skl_bgc_out, solve_zsal_out, grid_o_out, l_sk_out, &
   initbio_frac_out, grid_oS_out, l_skS_out, &
   phi_snow_out, conserv_check_out, &
   fr_resp_out, algal_vel_out, R_dFe2dust_out, dustFe_sol_out, &
   T_max_out, fsal_out, op_dep_min_out, fr_graze_s_out, fr_graze_e_out, &
   fr_mort2min_out, fr_resp_s_out, fr_dFe_out, &
   k_nitrif_out, t_iron_conv_out, max_loss_out, max_dfe_doc1_out, &
   y_sk_DMS_out, t_sk_conv_out, t_sk_ox_out, frazil_scav_out, &
   sw_redist_out, sw_frac_out, sw_dtemp_out, snwgrain_out, &
   snwredist_out, use_smliq_pnd_out, rsnw_fall_out, rsnw_tmax_out, &
   rhosnew_out, rhosmin_out, rhosmax_out, windmin_out, drhosdwind_out, &
   snwlvlfac_out, isnw_T_out, isnw_Tgrd_out, isnw_rhos_out, &
   snowage_rhos_out, snowage_Tgrd_out, snowage_T_out, &
   snowage_tau_out, snowage_kappa_out, snowage_drdt0_out, &
   snw_aging_table_out, snw_ssp_table_out )

!-----------------------------------------------------------------
! control settings
!-----------------------------------------------------------------

character(len=*), intent(out), optional :: &
   argcheck_out      ! optional argument checking

!-----------------------------------------------------------------
! parameter constants
!-----------------------------------------------------------------

real (kind=dbl_kind), intent(out), optional :: &
   c0_out, c1_out, c1p5_out, c2_out, c3_out, c4_out, &
   c5_out, c6_out, c8_out, c10_out, c15_out, c16_out, c20_out, &
   c25_out, c180_out, c100_out, c1000_out, p001_out, p01_out, p1_out, &
   p2_out, p4_out, p5_out, p6_out, p05_out, p15_out, p25_out, p75_out, &
   p333_out, p666_out, spval_const_out, pih_out, piq_out, pi2_out, &
   secday_out,     & ! number of seconds per day
   puny_out,       & ! a small number
   bignum_out,     & ! a big number
   pi_out,         & ! pi
   rad_to_deg_out, & ! conversion factor from radians to degrees
   Lfresh_out,     & ! latent heat of melting of fresh ice (J/kg)
   cprho_out,      & ! for ocean mixed layer (J kg / K m^3)
   Cp_out            ! proport const for PE

!-----------------------------------------------------------------
! densities
!-----------------------------------------------------------------

real (kind=dbl_kind), intent(out), optional :: &
   rhos_out,       & ! density of snow (kg/m^3)
   rhoi_out,       & ! density of ice (kg/m^3)
   rhosi_out,      & ! average sea ice density (kg/m2)
```

(continues on next page)

(continued from previous page)

```
        rhow_out,       & ! density of seawater (kg/m^3)
        rhofresh_out      ! density of fresh water (kg/m^3)

!-----------------------------------------------------------------------
! Parameters for thermodynamics
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(out), optional :: &
         floediam_out,   & ! effective floe diameter for lateral melt (m)
         hfrazilmin_out, & ! min thickness of new frazil ice (m)
         cp_ice_out,     & ! specific heat of fresh ice (J/kg/K)
         cp_ocn_out,     & ! specific heat of ocn    (J/kg/K)
         depressT_out,   & ! Tf:brine salinity ratio (C/ppt)
         viscosity_dyn_out, & ! dynamic viscosity of brine (kg/m/s)
         tscale_pnd_drain_out, & ! mushy macroscopic drainage timescale (days)
         Tocnfrz_out,    & ! freezing temp of seawater (C)
         Tffresh_out,    & ! freezing temp of fresh ice (K)
         Lsub_out,       & ! latent heat, sublimation freshwater (J/kg)
         Lvap_out,       & ! latent heat, vaporization freshwater (J/kg)
         Timelt_out,     & ! melting temperature, ice top surface  (C)
         Tsmelt_out,     & ! melting temperature, snow top surface (C)
         ice_ref_salinity_out, & ! (ppt)
         kice_out,       & ! thermal conductivity of fresh ice(W/m/deg)
         ksno_out,       & ! thermal conductivity of snow  (W/m/deg)
         hs_min_out,     & ! min snow thickness for computing zTsn (m)
         snowpatch_out,  & ! parameter for fractional snow area (m)
         saltmax_out,    & ! max salinity at ice base for BL99 (ppt)
         phi_init_out,   & ! initial liquid fraction of frazil
         min_salin_out,  & ! threshold for brine pocket treatment
         salt_loss_out,  & ! fraction of salt retained in zsalinity
         Tliquidus_max_out, & ! maximum liquidus temperature of mush (C)
         dSin0_frazil_out  ! bulk salinity reduction of newly formed frazil

      integer (kind=int_kind), intent(out), optional :: &
         ktherm_out         ! type of thermodynamics
                            ! -1 none
                            ! 1 = Bitz and Lipscomb 1999
                            ! 2 = mushy layer theory

      character (len=*), intent(out), optional :: &
         conduct_out, &        ! 'MU71' or 'bubbly'
         fbot_xfer_type_out, & ! transfer coefficient type for ice-ocean heat flux
         cpl_frazil_out        ! type of coupling for frazil ice

      logical (kind=log_kind), intent(out), optional :: &
         calc_Tsfc_out    ,&! if true, calculate surface temperature
                            ! if false, Tsfc is computed elsewhere and
                            ! atmos-ice fluxes are provided to CICE
         update_ocn_f_out   ! include fresh water and salt fluxes for frazil

      real (kind=dbl_kind), intent(out), optional :: &
         dts_b_out,   &      ! zsalinity timestep
```

(continues on next page)

(continued from previous page)

```
         hi_min_out,  &      ! minimum ice thickness allowed (m) for thermo
         ustar_min_out       ! minimum friction velocity for ice-ocean heat flux

      ! mushy thermo
      real(kind=dbl_kind), intent(out), optional :: &
         a_rapid_mode_out      , & ! channel radius for rapid drainage mode (m)
         Rac_rapid_mode_out    , & ! critical Rayleigh number for rapid drainage mode
         aspect_rapid_mode_out , & ! aspect ratio for rapid drainage mode (larger=wider)
         dSdt_slow_mode_out    , & ! slow mode drainage strength (m s-1 K-1)
         phi_c_slow_mode_out   , & ! liquid fraction porosity cutoff for slow mode
         phi_i_mushy_out           ! liquid fraction of congelation ice

      character(len=*), intent(out), optional :: &
         tfrz_option_out              ! form of ocean freezing temperature
                                      ! 'minus1p8' = -1.8 C
                                      ! 'constant' = Tocnfrz
                                      ! 'linear_salt' = -depressT * sss
                                      ! 'mushy' conforms with ktherm=2

      character(len=*), intent(out), optional :: &
         saltflux_option_out         ! Salt flux computation
                                     ! 'constant' reference value of ice_ref_salinity
                                     ! 'prognostic' prognostic salt flux


!-----------------------------------------------------------------------
! Parameters for radiation
!-----------------------------------------------------------------------

      real(kind=dbl_kind), intent(out), optional :: &
         emissivity_out, & ! emissivity of snow and ice
         albocn_out,     & ! ocean albedo
         vonkar_out,     & ! von Karman constant
         stefan_boltzmann_out, & !  W/m^2/K^4
         kappav_out,     & ! vis extnctn coef in ice, wvlngth<700nm (1/m)
         hi_ssl_out,     & ! ice surface scattering layer thickness (m)
         hs_ssl_out,     & ! visible, direct
         awtvdr_out,     & ! visible, direct  ! for history and
         awtidr_out,     & ! near IR, direct  ! diagnostics
         awtvdf_out,     & ! visible, diffuse
         awtidf_out          ! near IR, diffuse

      character (len=*), intent(out), optional :: &
         shortwave_out, & ! shortwave method, 'ccsm3' or 'dEdd' or 'dEdd_snicar_ad'
         albedo_type_out  ! albedo parameterization, 'ccsm3' or 'constant'
                            ! shortwave='dEdd' overrides this parameter

      ! baseline albedos for ccsm3 shortwave, set in namelist
      real (kind=dbl_kind), intent(out), optional :: &
         albicev_out  , & ! visible ice albedo for h > ahmax
         albicei_out  , & ! near-ir ice albedo for h > ahmax
         albsnowv_out , & ! cold snow albedo, visible
```

(continues on next page)

(continued from previous page)

```
         albsnowi_out , & ! cold snow albedo, near IR
         ahmax_out        ! thickness above which ice albedo is constant (m)

      ! dEdd tuning parameters, set in namelist
      real (kind=dbl_kind), intent(out), optional :: &
         R_ice_out    , & ! sea ice tuning parameter; +1 > 1sig increase in albedo
         R_pnd_out    , & ! ponded ice tuning parameter; +1 > 1sig increase in albedo
         R_snw_out    , & ! snow tuning parameter; +1 > ~.01 change in broadband albedo
         dT_mlt_out   , & ! change in temp for non-melt to melt snow grain
                          ! radius change (C)
         rsnw_mlt_out , & ! maximum melting snow grain radius (10^-6 m)
         kalg_out         ! algae absorption coefficient for 0.5 m thick layer

      logical (kind=log_kind), intent(out), optional :: &
         sw_redist_out    ! redistribute shortwave

      real (kind=dbl_kind), intent(out), optional :: &
         sw_frac_out  , & ! Fraction of internal shortwave moved to surface
         sw_dtemp_out     ! temperature difference from melting

!-----------------------------------------------------------------------
! Parameters for dynamics
!-----------------------------------------------------------------------

      real(kind=dbl_kind), intent(out), optional :: &
         Cf_out,         & ! ratio of ridging work to PE change in ridging
         Pstar_out,      & ! constant in Hibler strength formula
         Cstar_out,      & ! constant in Hibler strength formula
         dragio_out,     & ! ice-ocn drag coefficient
         thickness_ocn_layer1_out, & ! thickness of first ocean level (m)
         iceruf_ocn_out, & ! under-ice roughness (m)
         gravit_out,     & ! gravitational acceleration (m/s^2)
         iceruf_out        ! ice surface roughness (m)

      integer (kind=int_kind), intent(out), optional :: & ! defined in namelist
         kstrength_out   , & ! 0 for simple Hibler (1979) formulation
                             ! 1 for Rothrock (1975) pressure formulation
         krdg_partic_out, & ! 0 for Thorndike et al. (1975) formulation
                             ! 1 for exponential participation function
         krdg_redist_out     ! 0 for Hibler (1980) formulation
                             ! 1 for exponential redistribution function

      real (kind=dbl_kind), intent(out), optional :: &
         mu_rdg_out          ! gives e-folding scale of ridged ice (m^.5)
                             ! (krdg_redist = 1)

      logical (kind=log_kind), intent(out), optional :: &
         calc_dragio_out     ! if true, compute dragio from iceruf_ocn and thickness_ocn_
↪layer1

!-----------------------------------------------------------------------
! Parameters for atmosphere
```

(continues on next page)

(continued from previous page)

```
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(out), optional :: &
         cp_air_out,     & ! specific heat of air (J/kg/K)
         cp_wv_out,      & ! specific heat of water vapor (J/kg/K)
         zvir_out,       & ! rh2o/rair - 1.0
         zref_out,       & ! reference height for stability (m)
         qqqice_out,     & ! for qsat over ice
         TTTice_out,     & ! for qsat over ice
         qqqocn_out,     & ! for qsat over ocn
         TTTocn_out        ! for qsat over ocn

      character (len=*), intent(out), optional :: &
         atmbndy_out    ! atmo boundary method, 'similarity', 'constant' or 'mixed'

      logical (kind=log_kind), intent(out), optional :: &
         calc_strair_out,     & ! if true, calculate wind stress components
         formdrag_out,        & ! if true, calculate form drag
         highfreq_out           ! if true, use high frequency coupling

      integer (kind=int_kind), intent(out), optional :: &
         natmiter_out        ! number of iterations for boundary layer calculations

      ! Flux convergence tolerance
      real (kind=dbl_kind), intent(out), optional :: atmiter_conv_out

!-----------------------------------------------------------------------
! Parameters for the ice thickness distribution
!-----------------------------------------------------------------------

      integer (kind=int_kind), intent(out), optional :: &
         kitd_out        , & ! type of itd conversions
                             !   0 = delta function
                             !   1 = linear remap
         kcatbound_out       !   0 = old category boundary formula
                             !   1 = new formula giving round numbers
                             !   2 = WMO standard
                             !   3 = asymptotic formula

!-----------------------------------------------------------------------
! Parameters for the floe size distribution
!-----------------------------------------------------------------------

      integer (kind=int_kind), intent(out), optional :: &
         nfreq_out          ! number of frequencies

      real (kind=dbl_kind), intent(out), optional :: &
         floeshape_out      ! constant from Steele (unitless)

      logical (kind=log_kind), intent(out), optional :: &
         wave_spec_out      ! if true, use wave forcing
```

(continues on next page)

(continued from previous page)

```
      character (len=*), intent(out), optional :: &
         wave_spec_type_out ! type of wave spectrum forcing

!-----------------------------------------------------------------------
! Parameters for biogeochemistry
!-----------------------------------------------------------------------

      character (len=*), intent(out), optional :: &
         bgc_flux_type_out    ! type of ocean-ice piston velocity
                              ! 'constant', 'Jin2006'

      logical (kind=log_kind), intent(out), optional :: &
         z_tracers_out,      & ! if .true., bgc or aerosol tracers are vertically␣
↪resolved
         scale_bgc_out,      & ! if .true., initialize bgc tracers proportionally with␣
↪salinity
         solve_zbgc_out,     & ! if .true., solve vertical biochemistry portion of code
         dEdd_algae_out,     & ! if .true., algal absorptionof Shortwave is computed in␣
↪the
         modal_aero_out,     & ! if .true., use modal aerosol formulation in shortwave
         conserv_check_out     ! if .true., run conservation checks and abort if checks␣
↪fail

      logical (kind=log_kind), intent(out), optional :: &
         skl_bgc_out,        &   ! if true, solve skeletal biochemistry
         solve_zsal_out          ! if true, update salinity profile from solve_S_dt

      real (kind=dbl_kind), intent(out), optional :: &
         grid_o_out      , & ! for bottom flux
         l_sk_out        , & ! characteristic diffusive scale (zsalinity) (m)
         initbio_frac_out, & ! fraction of ocean tracer concentration used to initialize␣
↪tracer
         phi_snow_out        ! snow porosity at the ice/snow interface

      real (kind=dbl_kind), intent(out), optional :: &
         grid_oS_out     , & ! for bottom flux (zsalinity)
         l_skS_out           ! 0.02 characteristic skeletal layer thickness (m)␣
↪(zsalinity)
      real (kind=dbl_kind), intent(out), optional :: &
         fr_resp_out            , &   ! fraction of algal growth lost due to respiration
         algal_vel_out          , &   ! 0.5 cm/d(m/s) Lavoie 2005  1.5 cm/day
         R_dFe2dust_out         , &   !  g/g (3.5% content) Tagliabue 2009
         dustFe_sol_out         , &   ! solubility fraction
         T_max_out             , & ! maximum temperature (C)
         fsal_out              , & ! Salinity limitation (ppt)
         op_dep_min_out        , & ! Light attenuates for optical depths exceeding min
         fr_graze_s_out        , & ! fraction of grazing spilled or slopped
         fr_graze_e_out        , & ! fraction of assimilation excreted
         fr_mort2min_out       , & ! fractionation of mortality to Am
         fr_dFe_out            , & ! fraction of remineralized nitrogen
                                   ! (in units of algal iron)
         k_nitrif_out          , & ! nitrification rate (1/day)
```

(continues on next page)

(continued from previous page)

```
         t_iron_conv_out      , & ! desorption loss pFe to dFe (day)
         max_loss_out         , & ! restrict uptake to % of remaining value
         max_dfe_doc1_out     , & ! max ratio of dFe to saccharides in the ice
                                    ! (nM Fe/muM C)
         fr_resp_s_out        , & ! DMSPd fraction of respiration loss as DMSPd
         y_sk_DMS_out         , & ! fraction conversion given high yield
         t_sk_conv_out        , & ! Stefels conversion time (d)
         t_sk_ox_out          , & ! DMS oxidation time (d)
         frazil_scav_out          ! scavenging fraction or multiple in frazil ice

      real (kind=dbl_kind), intent(out), optional :: &
         sk_l_out,       & ! skeletal layer thickness (m)
         min_bgc_out       ! fraction of ocean bgc concentration in surface melt

!-----------------------------------------------------------------------
! Parameters for melt ponds
!-----------------------------------------------------------------------

      real (kind=dbl_kind), intent(out), optional :: &
         hs0_out             ! snow depth for transition to bare sea ice (m)

      ! level-ice ponds
      character (len=*), intent(out), optional :: &
         frzpnd_out          ! pond refreezing parameterization

      real (kind=dbl_kind), intent(out), optional :: &
         dpscale_out, &      ! alter e-folding time scale for flushing
         rfracmin_out, &     ! minimum retained fraction of meltwater
         rfracmax_out, &     ! maximum retained fraction of meltwater
         pndaspect_out, &    ! ratio of pond depth to pond fraction
         hs1_out             ! tapering parameter for snow on pond ice

      ! topo ponds
      real (kind=dbl_kind), intent(out), optional :: &
         hp1_out             ! critical parameter for pond ice thickness

!-----------------------------------------------------------------------
! Parameters for snow redistribution, metamorphosis
!-----------------------------------------------------------------------

      character (len=*), intent(out), optional :: &
         snwredist_out, &    ! type of snow redistribution
         snw_aging_table_out ! snow aging lookup table

      logical (kind=log_kind), intent(out), optional :: &
         use_smliq_pnd_out, &! use liquid in snow for ponds
         snwgrain_out        ! snow metamorphosis

      real (kind=dbl_kind), intent(out), optional :: &
         rsnw_fall_out, &    ! radius of new snow (10^-6 m)
         rsnw_tmax_out, &    ! maximum snow radius (10^-6 m)
         rhosnew_out, &      ! new snow density (kg/m^3)
```

(continues on next page)

(continued from previous page)

```
         rhosmin_out, &       ! minimum snow density (kg/m^3)
         rhosmax_out, &       ! maximum snow density (kg/m^3)
         windmin_out, &       ! minimum wind speed to compact snow (m/s)
         drhosdwind_out, &    ! wind compaction factor (kg s/m^4)
         snwlvlfac_out        ! fractional increase in snow depth

      integer (kind=int_kind), intent(out), optional :: &
         isnw_T_out, &        ! maxiumum temperature index
         isnw_Tgrd_out, &     ! maxiumum temperature gradient index
         isnw_rhos_out        ! maxiumum snow density index

      real (kind=dbl_kind), dimension(:), intent(out), optional :: &
         snowage_rhos_out, & ! snowage dimension data
         snowage_Tgrd_out, & !
         snowage_T_out       !

      real (kind=dbl_kind), dimension(:,:,:), intent(out), optional :: &
         snowage_tau_out, &  ! (10^-6 m)
         snowage_kappa_out, &!
         snowage_drdt0_out   ! (10^-6 m/hr)

      character (len=char_len), intent(out), optional :: &
         snw_ssp_table_out   ! lookup table: 'snicar' or 'test'
```

icepack_write_parameters

```
! subroutine to write the column package internal parameters

      subroutine icepack_write_parameters(iounit)

        integer (kind=int_kind), intent(in) :: &
             iounit    ! unit number for output
```

icepack_recompute_constants

```
! subroutine to reinitialize some derived constants

      subroutine icepack_recompute_constants()
```

4.4.11 icepack_shortwave.F90

icepack_init_radiation

```
! Initialize data needed for shortwave radiation calculations
! This should be called after values are set via icepack_init_parameters

      subroutine icepack_init_radiation()
```

icepack_prep_radiation

```
! Scales radiation fields computed on the previous time step.
!
! authors: Elizabeth Hunke, LANL

      subroutine icepack_prep_radiation(aice,        aicen,    &
                                        swvdr,       swvdf,    &
                                        swidr,       swidf,    &
                                        alvdr_ai,    alvdf_ai, &
                                        alidr_ai,    alidf_ai, &
                                        scale_factor,          &
                                        fswsfcn,     fswintn,  &
                                        fswthrun,              &
                                        fswthrun_vdr,          &
                                        fswthrun_vdf,          &
                                        fswthrun_idr,          &
                                        fswthrun_idf,          &
                                        fswpenln,              &
                                        Sswabsn,     Iswabsn)

      real (kind=dbl_kind), intent(in) :: &
         aice        , & ! ice area fraction
         swvdr       , & ! sw down, visible, direct  (W/m^2)
         swvdf       , & ! sw down, visible, diffuse (W/m^2)
         swidr       , & ! sw down, near IR, direct  (W/m^2)
         swidf       , & ! sw down, near IR, diffuse (W/m^2)
         ! grid-box-mean albedos aggregated over categories (if calc_Tsfc)
         alvdr_ai    , & ! visible, direct   (fraction)
         alidr_ai    , & ! near-ir, direct   (fraction)
         alvdf_ai    , & ! visible, diffuse  (fraction)
         alidf_ai        ! near-ir, diffuse  (fraction)

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         aicen           ! ice area fraction in each category

      real (kind=dbl_kind), intent(inout) :: &
         scale_factor    ! shortwave scaling factor, ratio new:old

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         fswsfcn     , & ! SW absorbed at ice/snow surface (W m-2)
         fswintn     , & ! SW absorbed in ice interior, below surface (W m-2)
```

(continues on next page)

(continued from previous page)

```
         fswthrun        ! SW through ice to ocean (W/m^2)

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         fswthrun_vdr , & ! vis dir SW through ice to ocean (W/m^2)
         fswthrun_vdf , & ! vis dif SW through ice to ocean (W/m^2)
         fswthrun_idr , & ! nir dir SW through ice to ocean (W/m^2)
         fswthrun_idf     ! nir dif SW through ice to ocean (W/m^2)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         fswpenln    , & ! visible SW entering ice layers (W m-2)
         Iswabsn     , & ! SW radiation absorbed in ice layers (W m-2)
         Sswabsn         ! SW radiation absorbed in snow layers (W m-2)
```

icepack_step_radiation

```
! Computes radiation fields
!
! authors: William H. Lipscomb, LANL
!          David Bailey, NCAR
!          Elizabeth C. Hunke, LANL

      subroutine icepack_step_radiation (dt,                 &
                                swgrid,   igrid,     &
                                fbri,                &
                                aicen,    vicen,     &
                                vsnon,    Tsfcn,     &
                                alvln,    apndn,     &
                                hpndn,    ipndn,     &
                                aeron,               &
                                bgcNn,    zaeron,    &
                                trcrn_bgcsw,         &
                                TLAT,     TLON,      &
                                calendar_type,       &
                                days_per_year,       &
                                nextsw_cday,         &
                                yday,     sec,       &
                                swvdr,    swvdf,     &
                                swidr,    swidf,     &
                                coszen,   fsnow,     &
                                alvdrn,   alvdfn,    &
                                alidrn,   alidfn,    &
                                fswsfcn,  fswintn,   &
                                fswthrun,            &
                                fswthrun_vdr,        &
                                fswthrun_vdf,        &
                                fswthrun_idr,        &
                                fswthrun_idf,        &
                                fswpenln,            &
                                Sswabsn,  Iswabsn,   &
                                albicen,  albsnon,   &
```

(continues on next page)

(continued from previous page)

```
                                        albpndn,  apeffn,    &
                                        snowfracn,           &
                                        dhsn,     ffracn,    &
                                        rsnow,               &
                                        l_print_point,       &
                                        initonly)

      real (kind=dbl_kind), intent(in) :: &
         dt         , & ! time step (s)
         swvdr      , & ! sw down, visible, direct  (W/m^2)
         swvdf      , & ! sw down, visible, diffuse (W/m^2)
         swidr      , & ! sw down, near IR, direct  (W/m^2)
         swidf      , & ! sw down, near IR, diffuse (W/m^2)
         fsnow      , & ! snowfall rate (kg/m^2 s)
         TLAT, TLON     ! latitude and longitude (radian)

      integer (kind=int_kind), intent(in) :: &
         sec            ! elapsed seconds into date

      real (kind=dbl_kind), intent(in) :: &
         yday           ! day of the year

      character (len=char_len), intent(in), optional :: &
         calendar_type ! differentiates Gregorian from other calendars

      integer (kind=int_kind), intent(in), optional :: &
         days_per_year ! number of days in one year

      real (kind=dbl_kind), intent(in), optional :: &
         nextsw_cday    ! julian day of next shortwave calculation

      real (kind=dbl_kind), intent(inout) :: &
         coszen         ! cosine solar zenith angle, < 0 for sun below horizon

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         igrid          ! biology vertical interface points

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         swgrid         ! grid for ice tracers used in dEdd scheme

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         aicen      , & ! ice area fraction in each category
         vicen      , & ! ice volume in each category (m)
         vsnon      , & ! snow volume in each category (m)
         Tsfcn      , & ! surface temperature (deg C)
         alvln      , & ! level-ice area fraction
         apndn      , & ! pond area fraction
         hpndn      , & ! pond depth (m)
         ipndn      , & ! pond refrozen lid thickness (m)
         fbri            ! brine fraction

      real(kind=dbl_kind), dimension(:,:), intent(in) :: &
```

(continues on next page)

(continued from previous page)

```
         aeron     , & ! aerosols (kg/m^3)
         bgcNn     , & ! bgc Nit tracers
         zaeron        ! bgcz aero tracers

      real(kind=dbl_kind), dimension(:,:), intent(inout) :: &
         trcrn_bgcsw   ! zaerosols (kg/m^3) and chla (mg/m^3)

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         alvdrn    , & ! visible, direct  albedo (fraction)
         alidrn    , & ! near-ir, direct   (fraction)
         alvdfn    , & ! visible, diffuse  (fraction)
         alidfn    , & ! near-ir, diffuse  (fraction)
         fswsfcn   , & ! SW absorbed at ice/snow surface (W m-2)
         fswintn   , & ! SW absorbed in ice interior, below surface (W m-2)
         fswthrun  , & ! SW through ice to ocean (W/m^2)
         snowfracn , & ! snow fraction on each category
         dhsn      , & ! depth difference for snow on sea ice and pond ice
         ffracn    , & ! fraction of fsurfn used to melt ipond
                       ! albedo components for history
         albicen   , & ! bare ice
         albsnon   , & ! snow
         albpndn   , & ! pond
         apeffn        ! effective pond area used for radiation calculation

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         fswthrun_vdr , & ! vis dir SW through ice to ocean (W/m^2)
         fswthrun_vdf , & ! vis dif SW through ice to ocean (W/m^2)
         fswthrun_idr , & ! nir dir SW through ice to ocean (W/m^2)
         fswthrun_idf     ! nir dif SW through ice to ocean (W/m^2)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         fswpenln  , & ! visible SW entering ice layers (W m-2)
         Iswabsn   , & ! SW radiation absorbed in ice layers (W m-2)
         Sswabsn       ! SW radiation absorbed in snow layers (W m-2)

      logical (kind=log_kind), intent(in) :: &
         l_print_point ! flag for printing diagnostics

      real (kind=dbl_kind), dimension(:,:), intent(inout), optional :: &
         rsnow         ! snow grain radius tracer (10^-6 m)

      logical (kind=log_kind), optional :: &
         initonly      ! flag to indicate init only, default is false
```

4.4.12 icepack_snow.F90

icepack_init_snow

```
! Updates snow tracers
!
! authors: Elizabeth C. Hunke, LANL
!          Nicole Jeffery, LANL

      subroutine icepack_init_snow
```

icepack_step_snow

```
! Updates snow tracers
!
! authors: Elizabeth C. Hunke, LANL
!          Nicole Jeffery, LANL

      subroutine icepack_step_snow(dt,        nilyr,     &
                                   nslyr,     ncat,      &
                                   wind,      aice,      &
                                   aicen,     vicen,     &
                                   vsnon,     Tsfc,      &
                                   zqin1,     zSin1,     &
                                   zqsn,                 &
                                   alvl,      vlvl,      &
                                   smice,     smliq,     &
                                   rsnw,      rhos_cmpn, &
                                   fresh,     fhocn,     &
                                   fsloss,    fsnow)

      integer (kind=int_kind), intent(in) :: &
         nslyr, & ! number of snow layers
         nilyr, & ! number of ice  layers
         ncat     ! number of thickness categories

      real (kind=dbl_kind), intent(in) :: &
         dt     , & ! time step
         wind   , & ! wind speed (m/s)
         fsnow  , & ! snowfall rate (kg m-2 s-1)
         aice       ! ice area fraction

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         aicen, & ! ice area fraction
         vicen, & ! ice volume (m)
         Tsfc , & ! surface temperature (C)
         zqin1, & ! ice upper layer enthalpy
         zSin1, & ! ice upper layer salinity
         alvl,  & ! level ice area tracer
         vlvl     ! level ice volume tracer
```

(continues on next page)

(continued from previous page)

```
      real (kind=dbl_kind), intent(inout) :: &
         fresh    , & ! fresh water flux to ocean (kg/m^2/s)
         fhocn    , & ! net heat flux to ocean (W/m^2)
         fsloss       ! rate of snow loss to leads (kg/m^2/s)

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         vsnon    ! snow volume (m)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         zqsn     , & ! snow enthalpy (J/m^3)
         smice    , & ! tracer for mass of ice in snow (kg/m^3)
         smliq    , & ! tracer for mass of liquid in snow (kg/m^3)
         rsnw     , & ! snow grain radius (10^-6 m)
         rhos_cmpn    ! effective snow density: compaction (kg/m^3)
```

4.4.13 icepack_therm_itd.F90

icepack_step_therm2

```
! Driver for thermodynamic changes not needed for coupling:
! transport in thickness space, lateral growth and melting.
!
! authors: William H. Lipscomb, LANL
!          Elizabeth C. Hunke, LANL

      subroutine icepack_step_therm2 (dt, ncat, nltrcr,           &
                                     nilyr,        nslyr,         &
                                     hin_max,      nblyr,         &
                                     aicen,                       &
                                     vicen,        vsnon,         &
                                     aicen_init,   vicen_init,    &
                                     trcrn,                       &
                                     aice0,        aice,          &
                                     trcr_depend,                 &
                                     trcr_base,    n_trcr_strata, &
                                     nt_strata,                   &
                                     Tf,           sss,           &
                                     salinz,                      &
                                     rside,        meltl,         &
                                     fside,        wlat,          &
                                     frzmlt,       frazil,        &
                                     frain,        fpond,         &
                                     fresh,        fsalt,         &
                                     fhocn,        update_ocn_f,  &
                                     bgrid,        cgrid,         &
                                     igrid,        faero_ocn,     &
                                     first_ice,    fzsal,         &
                                     flux_bio,     ocean_bio,     &
                                     frazil_diag,                 &
                                     frz_onset,    yday,          &
```

(continues on next page)

(continued from previous page)

```
                              fiso_ocn,     HDO_ocn,       &
                              H2_16O_ocn,   H2_18O_ocn,    &
                              nfsd,         wave_sig_ht,   &
                              wave_spectrum,               &
                              wavefreq,                    &
                              dwavefreq,                   &
                              d_afsd_latg,  d_afsd_newi,   &
                              d_afsd_latm,  d_afsd_weld,   &
                              floe_rad_c,   floe_binwidth)

    use icepack_parameters, only: icepack_init_parameters

    integer (kind=int_kind), intent(in) :: &
       ncat     , & ! number of thickness categories
       nltrcr   , & ! number of zbgc tracers
       nblyr    , & ! number of bio layers
       nilyr    , & ! number of ice layers
       nslyr        ! number of snow layers

    integer (kind=int_kind), intent(in), optional :: &
       nfsd         ! number of floe size categories

    logical (kind=log_kind), intent(in), optional :: &
       update_ocn_f ! if true, update fresh water and salt fluxes

    real (kind=dbl_kind), dimension(0:ncat), intent(in) :: &
       hin_max      ! category boundaries (m)

    real (kind=dbl_kind), intent(in) :: &
       dt       , & ! time step
       Tf       , & ! freezing temperature (C)
       sss      , & ! sea surface salinity (ppt)
       rside    , & ! fraction of ice that melts laterally
       frzmlt       ! freezing/melting potential (W/m^2)

    integer (kind=int_kind), dimension (:), intent(in) :: &
       trcr_depend, & ! = 0 for aicen tracers, 1 for vicen, 2 for vsnon
       n_trcr_strata  ! number of underlying tracer layers

    real (kind=dbl_kind), dimension (:,:), intent(in) :: &
       trcr_base    ! = 0 or 1 depending on tracer dependency
                    ! argument 2:  (1) aice, (2) vice, (3) vsno

    integer (kind=int_kind), dimension (:,:), intent(in) :: &
       nt_strata    ! indices of underlying tracer layers

    real (kind=dbl_kind), dimension (nblyr+2), intent(in) :: &
       bgrid        ! biology nondimensional vertical grid points

    real (kind=dbl_kind), dimension (nblyr+1), intent(in) :: &
       igrid        ! biology vertical interface points
```

(continues on next page)

(continued from previous page)

```
      real (kind=dbl_kind), dimension (nilyr+1), intent(in) :: &
         cgrid        ! CICE vertical coordinate

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         salinz    , & ! initial salinity profile
         ocean_bio     ! ocean concentration of biological tracer

      real (kind=dbl_kind), intent(inout) :: &
         aice      , & ! sea ice concentration
         aice0     , & ! concentration of open water
         fside     , & ! lateral heat flux (W/m^2)
         frain     , & ! rainfall rate (kg/m^2 s)
         fpond     , & ! fresh water flux to ponds (kg/m^2/s)
         fresh     , & ! fresh water flux to ocean (kg/m^2/s)
         fsalt     , & ! salt flux to ocean (kg/m^2/s)
         fhocn     , & ! net heat flux to ocean (W/m^2)
         meltl     , & ! lateral ice melt         (m/step-->cm/day)
         frazil    , & ! frazil ice growth        (m/step-->cm/day)
         frazil_diag   ! frazil ice growth diagnostic (m/step-->cm/day)

      real (kind=dbl_kind), intent(inout), optional :: &
         fzsal        ! salt flux to ocean from zsalinity (kg/m^2/s) (deprecated)

      real (kind=dbl_kind), intent(in), optional :: &
         wlat         ! lateral melt rate (m/s)

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         aicen_init,& ! initial concentration of ice
         vicen_init,& ! initial volume per unit area of ice          (m)
         aicen     , & ! concentration of ice
         vicen     , & ! volume per unit area of ice          (m)
         vsnon     , & ! volume per unit area of snow         (m)
         faero_ocn, & ! aerosol flux to ocean  (kg/m^2/s)
         flux_bio     ! all bio fluxes to ocean

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         trcrn        ! tracers

      logical (kind=log_kind), dimension(:), intent(inout) :: &
         first_ice    ! true until ice forms

      real (kind=dbl_kind), intent(inout), optional :: &
         frz_onset    ! day of year that freezing begins (congel or frazil)

      real (kind=dbl_kind), intent(in), optional :: &
         yday         ! day of year

      ! water isotopes
      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         fiso_ocn     ! isotope flux to ocean  (kg/m^2/s)

      real (kind=dbl_kind), intent(in), optional :: &
```

(continues on next page)

(continued from previous page)

```
         HDO_ocn    , & ! ocean concentration of HDO (kg/kg)
         H2_16O_ocn , & ! ocean concentration of H2_16O (kg/kg)
         H2_18O_ocn     ! ocean concentration of H2_18O (kg/kg)

      real (kind=dbl_kind), intent(in), optional :: &
         wave_sig_ht    ! significant height of waves in ice (m)

      real (kind=dbl_kind), dimension(:), intent(in), optional  :: &
         wave_spectrum  ! ocean surface wave spectrum E(f) (m^2 s)

      real(kind=dbl_kind), dimension(:), intent(in), optional :: &
         wavefreq, &    ! wave frequencies (s^-1)
         dwavefreq      ! wave frequency bin widths (s^-1)

      real (kind=dbl_kind), dimension(:), intent(out), optional :: &
                        ! change in floe size distribution (area)
         d_afsd_latg, & ! due to fsd lateral growth
         d_afsd_newi, & ! new ice formation
         d_afsd_latm, & ! lateral melt
         d_afsd_weld    ! welding

      real (kind=dbl_kind), dimension (:), intent(in), optional :: &
         floe_rad_c, &  ! fsd size bin centre in m (radius)
         floe_binwidth  ! fsd size bin width in m (radius)
```

4.4.14 icepack_therm_shared.F90

icepack_init_thermo

```
! Initialize the vertical profile of ice salinity and melting temperature.
!
! authors: C. M. Bitz, UW
!          William H. Lipscomb, LANL

      subroutine icepack_init_thermo(nilyr, sprofile)

      integer (kind=int_kind), intent(in) :: &
         nilyr                            ! number of ice layers

      real (kind=dbl_kind), dimension(:), intent(out) :: &
         sprofile                         ! vertical salinity profile
```

icepack_salinity_profile

```
! Initial salinity profile
!
! authors: C. M. Bitz, UW
!          William H. Lipscomb, LANL

      function icepack_salinity_profile(zn) result(salinity)

      real(kind=dbl_kind), intent(in) :: &
         zn ! depth

      real(kind=dbl_kind) :: &
         salinity ! initial salinity profile
```

icepack_init_trcr

```
!
      subroutine icepack_init_trcr(Tair,     Tf,       &
                                   Sprofile, Tprofile, &
                                   Tsfc,               &
                                   nilyr,    nslyr,    &
                                   qin,      qsn)

      integer (kind=int_kind), intent(in) :: &
         nilyr, &    ! number of ice layers
         nslyr       ! number of snow layers

      real (kind=dbl_kind), intent(in) :: &
         Tair, &     ! air temperature (K)
         Tf          ! freezing temperature (C)

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         Sprofile, & ! vertical salinity profile (ppt)
         Tprofile    ! vertical temperature profile (C)

      real (kind=dbl_kind), intent(out) :: &
         Tsfc        ! surface temperature (C)

      real (kind=dbl_kind), dimension(:), intent(out) :: &
         qin, &      ! ice enthalpy profile (J/m3)
         qsn         ! snow enthalpy profile (J/m3)
```

icepack_liquidus_temperature

```
! compute liquidus temperature

      function icepack_liquidus_temperature(Sin) result(Tmlt)

        real(dbl_kind), intent(in) :: Sin
        real(dbl_kind) :: Tmlt
```

icepack_sea_freezing_temperature

```
! compute ocean freezing temperature

      function icepack_sea_freezing_temperature(sss) result(Tf)

        real(dbl_kind), intent(in) :: sss
        real(dbl_kind) :: Tf
```

icepack_ice_temperature

```
! compute ice temperature

      function icepack_ice_temperature(qin, Sin) result(Tin)

        real(kind=dbl_kind), intent(in) :: qin, Sin
        real(kind=dbl_kind) :: Tin
```

icepack_snow_temperature

```
! compute snow temperature

      function icepack_snow_temperature(qin) result(Tsn)

        real(kind=dbl_kind), intent(in) :: qin
        real(kind=dbl_kind) :: Tsn
```

4.4.15 icepack_therm_vertical.F90

icepack_step_therm1

```
! Driver for thermodynamic changes not needed for coupling:
! transport in thickness space, lateral growth and melting.
!
! authors: William H. Lipscomb, LANL
!          Elizabeth C. Hunke, LANL
```

(continues on next page)

(continued from previous page)

```
      subroutine icepack_step_therm1(dt, ncat, nilyr, nslyr,    &
                                     aicen_init  ,              &
                                     vicen_init  , vsnon_init , &
                                     aice        , aicen      , &
                                     vice        , vicen      , &
                                     vsno        , vsnon      , &
                                     uvel        , vvel       , &
                                     Tsfc        , zqsn       , &
                                     zqin        , zSin       , &
                                     alvl        , vlvl       , &
                                     apnd        , hpnd       , &
                                     ipnd        ,              &
                                     iage        , FY         , &
                                     aerosno     , aeroice    , &
                                     isosno      , isoice     , &
                                     uatm        , vatm       , &
                                     wind        , zlvl       , &
                                     Qa          , rhoa       , &
                                     Qa_iso      , &
                                     Tair        , Tref       , &
                                     Qref        , Uref       , &
                                     Qref_iso    , &
                                     Cdn_atm_ratio,             &
                                     Cdn_ocn     , Cdn_ocn_skin, &
                                     Cdn_ocn_floe, Cdn_ocn_keel, &
                                     Cdn_atm     , Cdn_atm_skin, &
                                     Cdn_atm_floe, Cdn_atm_pond, &
                                     Cdn_atm_rdg , hfreebd    , &
                                     hdraft      , hridge     , &
                                     distrdg     , hkeel      , &
                                     dkeel       , lfloe      , &
                                     dfloe       ,              &
                                     strax       , stray      , &
                                     strairxT    , strairyT   , &
                                     potT        , sst        , &
                                     sss         , Tf         , &
                                     strocnxT    , strocnyT   , &
                                     fbot        ,              &
                                     Tbot        , Tsnice     , &
                                     frzmlt      , rside      , &
                                     fside       , wlat       , &
                                     fsnow       , frain      , &
                                     fpond       , fsloss     , &
                                     fsurf       , fsurfn     , &
                                     fcondtop    , fcondtopn  , &
                                     fcondbot    , fcondbotn  , &
                                     fswsfcn     , fswintn    , &
                                     fswthrun    ,              &
                                     fswthrun_vdr,              &
                                     fswthrun_vdf,              &
                                     fswthrun_idr,              &
                                     fswthrun_idf,              &
```

(continues on next page)

(continued from previous page)

```
                                    fswabs      ,               &
                                    flwout      ,               &
                                    Sswabsn     , Iswabsn     , &
                                    flw         , &
                                    fsens       , fsensn      , &
                                    flat        , flatn       , &
                                    evap        ,               &
                                    evaps       , evapi       , &
                                    fresh       , fsalt       , &
                                    fhocn       ,               &
                                    fswthru     ,               &
                                    fswthru_vdr ,               &
                                    fswthru_vdf ,               &
                                    fswthru_idr ,               &
                                    fswthru_idf ,               &
                                    flatn_f     , fsensn_f    , &
                                    fsurfn_f    , fcondtopn_f , &
                                    faero_atm   , faero_ocn   , &
                                    fiso_atm    , fiso_ocn    , &
                                    fiso_evap   , &
                                    HDO_ocn     , H2_16O_ocn  , &
                                    H2_18O_ocn  ,  &
                                    dhsn        , ffracn      , &
                                    meltt       , melttn      , &
                                    meltb       , meltbn      , &
                                    melts       , meltsn      , &
                                    congel      , congeln     , &
                                    snoice      , snoicen     , &
                                    dsnow       , dsnown      , &
                                    meltsliq    , meltsliqn   , &
                                    rsnwn       , &
                                    smicen      , smliqn      , &
                                    lmask_n     , lmask_s     , &
                                    mlt_onset   , frz_onset   , &
                                    yday        , prescribed_ice, &
                                    zlvs)

      integer (kind=int_kind), intent(in) :: &
         ncat        , & ! number of thickness categories
         nilyr       , & ! number of ice layers
         nslyr           ! number of snow layers

      real (kind=dbl_kind), intent(in) :: &
         dt          , & ! time step
         uvel        , & ! x-component of velocity              (m/s)
         vvel        , & ! y-component of velocity              (m/s)
         strax       , & ! wind stress components             (N/m^2)
         stray       , & !
         yday            ! day of year

      logical (kind=log_kind), intent(in) :: &
         lmask_n     , & ! northern hemisphere mask
```

(continues on next page)

(continued from previous page)

```
         lmask_s          ! southern hemisphere mask

      logical (kind=log_kind), intent(in), optional :: &
         prescribed_ice  ! if .true., use prescribed ice instead of computed

      real (kind=dbl_kind), intent(inout) :: &
         aice        , & ! sea ice concentration
         vice        , & ! volume per unit area of ice          (m)
         vsno        , & ! volume per unit area of snow         (m)
         zlvl        , & ! atm level height for momentum (and scalars if zlvs is not␣
↪present) (m)
         uatm        , & ! wind velocity components            (m/s)
         vatm        , & !                                     (m/s)
         wind        , & ! wind speed                          (m/s)
         potT        , & ! air potential temperature             (K)
         Tair        , & ! air temperature                       (K)
         Qa          , & ! specific humidity                 (kg/kg)
         rhoa        , & ! air density                      (kg/m^3)
         frain       , & ! rainfall rate                  (kg/m^2 s)
         fsnow       , & ! snowfall rate                  (kg/m^2 s)
         fpond       , & ! fresh water flux to ponds      (kg/m^2/s)
         fresh       , & ! fresh water flux to ocean      (kg/m^2/s)
         fsalt       , & ! salt flux to ocean             (kg/m^2/s)
         fhocn       , & ! net heat flux to ocean            (W/m^2)
         fswthru     , & ! shortwave penetrating to ocean    (W/m^2)
         fsurf       , & ! net surface heat flux (excluding fcondtop)(W/m^2)
         fcondtop    , & ! top surface conductive flux       (W/m^2)
         fcondbot    , & ! bottom surface conductive flux    (W/m^2)
         fsens       , & ! sensible heat flux                (W/m^2)
         flat        , & ! latent heat flux                  (W/m^2)
         fswabs      , & ! shortwave flux absorbed in ice and ocean (W/m^2)
         flw         , & ! incoming longwave radiation       (W/m^2)
         flwout      , & ! outgoing longwave radiation       (W/m^2)
         evap        , & ! evaporative water flux         (kg/m^2/s)
         evaps       , & ! evaporative water flux over snow(kg/m^2/s)
         evapi       , & ! evaporative water flux over ice (kg/m^2/s)
         congel      , & ! basal ice growth         (m/step-->cm/day)
         snoice      , & ! snow-ice formation       (m/step-->cm/day)
         Tref        , & ! 2m atm reference temperature          (K)
         Qref        , & ! 2m atm reference spec humidity    (kg/kg)
         Uref        , & ! 10m atm reference wind speed        (m/s)
         Cdn_atm     , & ! atm drag coefficient
         Cdn_ocn     , & ! ocn drag coefficient
         hfreebd     , & ! freeboard                             (m)
         hdraft      , & ! draft of ice + snow column (Stoessel1993)
         hridge      , & ! ridge height
         distrdg     , & ! distance between ridges
         hkeel       , & ! keel depth
         dkeel       , & ! distance between keels
         lfloe       , & ! floe length
         dfloe       , & ! distance between floes
         Cdn_atm_skin, & ! neutral skin drag coefficient
```

(continues on next page)

(continued from previous page)

```
         Cdn_atm_floe, & ! neutral floe edge drag coefficient
         Cdn_atm_pond, & ! neutral pond edge drag coefficient
         Cdn_atm_rdg , & ! neutral ridge drag coefficient
         Cdn_ocn_skin, & ! skin drag coefficient
         Cdn_ocn_floe, & ! floe edge drag coefficient
         Cdn_ocn_keel, & ! keel drag coefficient
         Cdn_atm_ratio,& ! ratio drag atm / neutral drag atm
         strairxT    , & ! stress on ice by air, x-direction
         strairyT    , & ! stress on ice by air, y-direction
         strocnxT    , & ! ice-ocean stress, x-direction
         strocnyT    , & ! ice-ocean stress, y-direction
         fbot        , & ! ice-ocean heat flux at bottom surface (W/m^2)
         frzmlt      , & ! freezing/melting potential         (W/m^2)
         rside       , & ! fraction of ice that melts laterally
         fside       , & ! lateral heat flux                  (W/m^2)
         sst         , & ! sea surface temperature                (C)
         Tf          , & ! freezing temperature                   (C)
         Tbot        , & ! ice bottom surface temperature     (deg C)
         Tsnice      , & ! snow ice interface temperature     (deg C)
         sss         , & ! sea surface salinity                 (ppt)
         meltt       , & ! top ice melt             (m/step-->cm/day)
         melts       , & ! snow melt                (m/step-->cm/day)
         meltb       , & ! basal ice melt           (m/step-->cm/day)
         mlt_onset   , & ! day of year that sfc melting begins
         frz_onset       ! day of year that freezing begins (congel or frazil)

      real (kind=dbl_kind), intent(out), optional :: &
         wlat            ! lateral melt rate                    (m/s)

      real (kind=dbl_kind), intent(inout), optional :: &
         fswthru_vdr , & ! vis dir shortwave penetrating to ocean (W/m^2)
         fswthru_vdf , & ! vis dif shortwave penetrating to ocean (W/m^2)
         fswthru_idr , & ! nir dir shortwave penetrating to ocean (W/m^2)
         fswthru_idf , & ! nir dif shortwave penetrating to ocean (W/m^2)
         dsnow       , & ! change in snow depth     (m/step-->cm/day)
         fsloss          ! rate of snow loss to leads      (kg/m^2/s)

      real (kind=dbl_kind), intent(out), optional :: &
         meltsliq        ! mass of snow melt                  (kg/m^2)

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         Qa_iso      , & ! isotope specific humidity          (kg/kg)
         Qref_iso    , & ! isotope 2m atm ref spec humidity   (kg/kg)
         fiso_atm    , & ! isotope deposition rate         (kg/m^2 s)
         fiso_ocn    , & ! isotope flux to ocean           (kg/m^2/s)
         fiso_evap       ! isotope evaporation             (kg/m^2/s)

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
         meltsliqn       ! mass of snow melt                  (kg/m^2)

      real (kind=dbl_kind), dimension(:,:), intent(inout), optional :: &
         rsnwn       , & ! snow grain radius                (10^-6 m)
```

(continues on next page)

(continued from previous page)

```
         smicen      , & ! tracer for mass of ice in snow    (kg/m^3)
         smliqn          ! tracer for mass of liq in snow    (kg/m^3)

      real (kind=dbl_kind), intent(in), optional :: &
         HDO_ocn     , & ! ocean concentration of HDO         (kg/kg)
         H2_16O_ocn  , & ! ocean concentration of H2_16O      (kg/kg)
         H2_18O_ocn  , & ! ocean concentration of H2_18O      (kg/kg)
         zlvs            ! atm level height for scalars (if different than zlvl) (m)

      real (kind=dbl_kind), dimension(:), intent(inout) :: &
         aicen_init  , & ! fractional area of ice
         vicen_init  , & ! volume per unit area of ice          (m)
         vsnon_init  , & ! volume per unit area of snow         (m)
         aicen       , & ! concentration of ice
         vicen       , & ! volume per unit area of ice          (m)
         vsnon       , & ! volume per unit area of snow         (m)
         Tsfc        , & ! ice/snow surface temperature, Tsfcn
         alvl        , & ! level ice area fraction
         vlvl        , & ! level ice volume fraction
         apnd        , & ! melt pond area fraction
         hpnd        , & ! melt pond depth                      (m)
         ipnd        , & ! melt pond refrozen lid thickness     (m)
         iage        , & ! volume-weighted ice age
         FY          , & ! area-weighted first-year ice area
         fsurfn      , & ! net flux to top surface, excluding fcondtop
         fcondtopn   , & ! downward cond flux at top surface  (W m-2)
         fcondbotn   , & ! downward cond flux at bottom surface (W m-2)
         flatn       , & ! latent heat flux                   (W m-2)
         fsensn      , & ! sensible heat flux                 (W m-2)
         fsurfn_f    , & ! net flux to top surface, excluding fcondtop
         fcondtopn_f , & ! downward cond flux at top surface  (W m-2)
         flatn_f     , & ! latent heat flux                   (W m-2)
         fsensn_f    , & ! sensible heat flux                 (W m-2)
         fswsfcn     , & ! SW absorbed at ice/snow surface    (W m-2)
         fswintn     , & ! SW absorbed in ice interior, below surface (W m-2)
         faero_atm   , & ! aerosol deposition rate         (kg/m^2 s)
         faero_ocn   , & ! aerosol flux to ocean           (kg/m^2/s)
         dhsn        , & ! depth difference for snow on sea ice and pond ice
         ffracn      , & ! fraction of fsurfn used to melt ipond
         meltsn      , & ! snow melt                            (m)
         melttn      , & ! top ice melt                         (m)
         meltbn      , & ! bottom ice melt                      (m)
         congeln     , & ! congelation ice growth               (m)
         snoicen     , & ! snow-ice growth                      (m)
         dsnown          ! change in snow thickness (m/step-->cm/day)

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         fswthrun        ! SW through ice to ocean            (W/m^2)

      real (kind=dbl_kind), dimension(:), intent(in), optional :: &
         fswthrun_vdr , & ! vis dir SW through ice to ocean   (W/m^2)
         fswthrun_vdf , & ! vis dif SW through ice to ocean   (W/m^2)
```

(continues on next page)

(continued from previous page)

```
         fswthrun_idr , & ! nir dir SW through ice to ocean   (W/m^2)
         fswthrun_idf     ! nir dif SW through ice to ocean   (W/m^2)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         zqsn        , & ! snow layer enthalpy                (J m-3)
         zqin        , & ! ice layer enthalpy                 (J m-3)
         zSin        , & ! internal ice layer salinities
         Sswabsn     , & ! SW radiation absorbed in snow layers (W m-2)
         Iswabsn         ! SW radiation absorbed in ice layers  (W m-2)

      real (kind=dbl_kind), dimension(:,:,:), intent(inout) :: &
         aerosno    , &  ! snow aerosol tracer               (kg/m^2)
         aeroice         ! ice aerosol tracer                (kg/m^2)

      real (kind=dbl_kind), dimension(:,:), intent(inout), optional :: &
         isosno     , &  ! snow isotope tracer               (kg/m^2)
         isoice          ! ice isotope tracer                (kg/m^2)
```

4.4.16 icepack_tracers.F90

icepack_init_tracer_flags

```
! set tracer active flags

      subroutine icepack_init_tracer_flags(&
           tr_iage_in, tr_FY_in, tr_lvl_in, tr_snow_in, &
           tr_pond_in, tr_pond_lvl_in, tr_pond_topo_in, &
           tr_fsd_in, tr_aero_in, tr_iso_in, tr_brine_in, tr_zaero_in, &
           tr_bgc_Nit_in, tr_bgc_N_in, tr_bgc_DON_in, tr_bgc_C_in, tr_bgc_chl_in, &
           tr_bgc_Am_in, tr_bgc_Sil_in, tr_bgc_DMS_in, tr_bgc_Fe_in, tr_bgc_hum_in, &
           tr_bgc_PON_in)

        logical, intent(in), optional :: &
             tr_iage_in      , & ! if .true., use age tracer
             tr_FY_in        , & ! if .true., use first-year area tracer
             tr_lvl_in       , & ! if .true., use level ice tracer
             tr_pond_in      , & ! if .true., use melt pond tracer
             tr_pond_lvl_in  , & ! if .true., use level-ice pond tracer
             tr_pond_topo_in , & ! if .true., use explicit topography-based ponds
             tr_snow_in      , & ! if .true., use snow redistribution or metamorphosis␣
↪tracers
             tr_fsd_in       , & ! if .true., use floe size distribution tracers
             tr_iso_in       , & ! if .true., use isotope tracers
             tr_aero_in      , & ! if .true., use aerosol tracers
             tr_brine_in     , & ! if .true., brine height differs from ice thickness
             tr_zaero_in     , & ! if .true., black carbon is tracers  (n_zaero)
             tr_bgc_Nit_in   , & ! if .true., Nitrate tracer in ice
             tr_bgc_N_in     , & ! if .true., algal nitrogen tracers  (n_algae)
             tr_bgc_DON_in   , & ! if .true., DON pools are tracers  (n_don)
             tr_bgc_C_in     , & ! if .true., algal carbon tracers + DOC and DIC
```

(continues on next page)

(continued from previous page)

```
             tr_bgc_chl_in   , & ! if .true., algal chlorophyll tracers
             tr_bgc_Am_in    , & ! if .true., ammonia/um as nutrient tracer
             tr_bgc_Sil_in   , & ! if .true., silicon as nutrient tracer
             tr_bgc_DMS_in   , & ! if .true., DMS as product tracer
             tr_bgc_Fe_in    , & ! if .true., Fe as product tracer
             tr_bgc_hum_in   , & ! if .true., hum as product tracer
             tr_bgc_PON_in       ! if .true., PON as product tracer
```

icepack_query_tracer_flags

```
! query tracer active flags

      subroutine icepack_query_tracer_flags(&
           tr_iage_out, tr_FY_out, tr_lvl_out, tr_snow_out, &
           tr_pond_out, tr_pond_lvl_out, tr_pond_topo_out, &
           tr_fsd_out, tr_aero_out, tr_iso_out, tr_brine_out, tr_zaero_out, &
           tr_bgc_Nit_out, tr_bgc_N_out, tr_bgc_DON_out, tr_bgc_C_out, tr_bgc_chl_out, &
           tr_bgc_Am_out, tr_bgc_Sil_out, tr_bgc_DMS_out, tr_bgc_Fe_out, tr_bgc_hum_out,␣
↪&
           tr_bgc_PON_out)

        logical, intent(out), optional :: &
             tr_iage_out      , & ! if .true., use age tracer
             tr_FY_out        , & ! if .true., use first-year area tracer
             tr_lvl_out       , & ! if .true., use level ice tracer
             tr_pond_out      , & ! if .true., use melt pond tracer
             tr_pond_lvl_out  , & ! if .true., use level-ice pond tracer
             tr_pond_topo_out , & ! if .true., use explicit topography-based ponds
             tr_snow_out      , & ! if .true., use snow redistribution or metamorphosis␣
↪tracers
             tr_fsd_out       , & ! if .true., use floe size distribution
             tr_iso_out       , & ! if .true., use isotope tracers
             tr_aero_out      , & ! if .true., use aerosol tracers
             tr_brine_out     , & ! if .true., brine height differs from ice thickness
             tr_zaero_out     , & ! if .true., black carbon is tracers  (n_zaero)
             tr_bgc_Nit_out   , & ! if .true., Nitrate tracer in ice
             tr_bgc_N_out     , & ! if .true., algal nitrogen tracers  (n_algae)
             tr_bgc_DON_out   , & ! if .true., DON pools are tracers  (n_don)
             tr_bgc_C_out     , & ! if .true., algal carbon tracers + DOC and DIC
             tr_bgc_chl_out   , & ! if .true., algal chlorophyll tracers
             tr_bgc_Am_out    , & ! if .true., ammonia/um as nutrient tracer
             tr_bgc_Sil_out   , & ! if .true., silicon as nutrient tracer
             tr_bgc_DMS_out   , & ! if .true., DMS as product tracer
             tr_bgc_Fe_out    , & ! if .true., Fe as product tracer
             tr_bgc_hum_out   , & ! if .true., hum as product tracer
             tr_bgc_PON_out       ! if .true., PON as product tracer
```

icepack_write_tracer_flags

```
! write tracer active flags

      subroutine icepack_write_tracer_flags(iounit)

        integer, intent(in) :: iounit
```

icepack_init_tracer_indices

```
! set the number of column tracer indices

      subroutine icepack_init_tracer_indices(&
           nt_Tsfc_in, nt_qice_in, nt_qsno_in, nt_sice_in, &
           nt_fbri_in, nt_iage_in, nt_FY_in, &
           nt_alvl_in, nt_vlvl_in, nt_apnd_in, nt_hpnd_in, nt_ipnd_in, &
           nt_smice_in, nt_smliq_in, nt_rhos_in, nt_rsnw_in, &
           nt_fsd_in, nt_isosno_in, nt_isoice_in, &
           nt_aero_in, nt_zaero_in, nt_bgc_C_in, &
           nt_bgc_N_in, nt_bgc_chl_in, nt_bgc_DOC_in, nt_bgc_DON_in, &
           nt_bgc_DIC_in, nt_bgc_Fed_in, nt_bgc_Fep_in, nt_bgc_Nit_in, nt_bgc_Am_in, &
           nt_bgc_Sil_in, nt_bgc_DMSPp_in, nt_bgc_DMSPd_in, nt_bgc_DMS_in, nt_bgc_hum_in,
↪ &
           nt_bgc_PON_in, nlt_zaero_in, nlt_bgc_C_in, nlt_bgc_N_in, nlt_bgc_chl_in, &
           nlt_bgc_DOC_in, nlt_bgc_DON_in, nlt_bgc_DIC_in, nlt_bgc_Fed_in, &
           nlt_bgc_Fep_in, nlt_bgc_Nit_in, nlt_bgc_Am_in, nlt_bgc_Sil_in, &
           nlt_bgc_DMSPp_in, nlt_bgc_DMSPd_in, nlt_bgc_DMS_in, nlt_bgc_hum_in, &
           nlt_bgc_PON_in, nt_zbgc_frac_in, nt_bgc_S_in, nlt_chl_sw_in, &
           nlt_zaero_sw_in, &
           bio_index_o_in, bio_index_in)

        integer, intent(in), optional :: &
             nt_Tsfc_in, & ! ice/snow temperature
             nt_qice_in, & ! volume-weighted ice enthalpy (in layers)
             nt_qsno_in, & ! volume-weighted snow enthalpy (in layers)
             nt_sice_in, & ! volume-weighted ice bulk salinity (CICE grid layers)
             nt_fbri_in, & ! volume fraction of ice with dynamic salt (hinS/vicen*aicen)
             nt_iage_in, & ! volume-weighted ice age
             nt_FY_in,   & ! area-weighted first-year ice area
             nt_alvl_in, & ! level ice area fraction
             nt_vlvl_in, & ! level ice volume fraction
             nt_apnd_in, & ! melt pond area fraction
             nt_hpnd_in, & ! melt pond depth
             nt_ipnd_in, & ! melt pond refrozen lid thickness
             nt_smice_in,& ! mass of ice in snow
             nt_smliq_in,& ! mass of liquid water in snow
             nt_rhos_in, & ! snow density
             nt_rsnw_in, & ! snow grain radius
             nt_fsd_in,  & ! floe size distribution
             nt_isosno_in,  & ! starting index for isotopes in snow
             nt_isoice_in,  & ! starting index for isotopes in ice
```

(continues on next page)

(continued from previous page)

```
         nt_aero_in,    & ! starting index for aerosols in ice
         nt_bgc_Nit_in, & ! nutrients
         nt_bgc_Am_in,  & !
         nt_bgc_Sil_in, & !
         nt_bgc_DMSPp_in,&! trace gases (skeletal layer)
         nt_bgc_DMSPd_in,&!
         nt_bgc_DMS_in, & !
         nt_bgc_hum_in, & !
         nt_bgc_PON_in, & ! zooplankton and detritus
         nlt_bgc_Nit_in,& ! nutrients
         nlt_bgc_Am_in, & !
         nlt_bgc_Sil_in,& !
         nlt_bgc_DMSPp_in,&! trace gases (skeletal layer)
         nlt_bgc_DMSPd_in,&!
         nlt_bgc_DMS_in,& !
         nlt_bgc_hum_in,& !
         nlt_bgc_PON_in,& ! zooplankton and detritus
         nt_zbgc_frac_in,&! fraction of tracer in the mobile phase
         nt_bgc_S_in,   & ! (deprecated, was related to zsalinity)
         nlt_chl_sw_in    ! points to total chla in trcrn_sw

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         bio_index_o_in, &
         bio_index_in

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         nt_bgc_N_in ,  & ! diatoms, phaeocystis, pico/small
         nt_bgc_C_in ,  & ! diatoms, phaeocystis, pico/small
         nt_bgc_chl_in, & ! diatoms, phaeocystis, pico/small
         nlt_bgc_N_in , & ! diatoms, phaeocystis, pico/small
         nlt_bgc_C_in , & ! diatoms, phaeocystis, pico/small
         nlt_bgc_chl_in   ! diatoms, phaeocystis, pico/small

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         nt_bgc_DOC_in, & !  dissolved organic carbon
         nlt_bgc_DOC_in   !  dissolved organic carbon

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         nt_bgc_DON_in, & !  dissolved organic nitrogen
         nlt_bgc_DON_in   !  dissolved organic nitrogen

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         nt_bgc_DIC_in, & ! dissolved inorganic carbon
         nlt_bgc_DIC_in   !  dissolved inorganic carbon

    integer (kind=int_kind), dimension(:), intent(in), optional :: &
         nt_bgc_Fed_in, & !  dissolved iron
         nt_bgc_Fep_in, & !  particulate iron
         nlt_bgc_Fed_in,& !  dissolved iron
         nlt_bgc_Fep_in   !  particulate iron

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

(continues on next page)

(continued from previous page)

```
             nt_zaero_in,   & !  black carbon and other aerosols
             nlt_zaero_in,  & !  black carbon and other aerosols
             nlt_zaero_sw_in  ! black carbon and dust in trcrn_sw
```

icepack_query_tracer_indices

```
! query the number of column tracer indices

      subroutine icepack_query_tracer_indices(&
           nt_Tsfc_out, nt_qice_out, nt_qsno_out, nt_sice_out, &
           nt_fbri_out, nt_iage_out, nt_FY_out, &
           nt_alvl_out, nt_vlvl_out, nt_apnd_out, nt_hpnd_out, nt_ipnd_out, &
           nt_smice_out, nt_smliq_out, nt_rhos_out, nt_rsnw_out, &
           nt_fsd_out, nt_isosno_out, nt_isoice_out, &
           nt_aero_out, nt_zaero_out, nt_bgc_C_out, &
           nt_bgc_N_out, nt_bgc_chl_out, nt_bgc_DOC_out, nt_bgc_DON_out, &
           nt_bgc_DIC_out, nt_bgc_Fed_out, nt_bgc_Fep_out, nt_bgc_Nit_out, nt_bgc_Am_out,
↪ &
           nt_bgc_Sil_out, nt_bgc_DMSPp_out, nt_bgc_DMSPd_out, nt_bgc_DMS_out, nt_bgc_
↪hum_out, &
           nt_bgc_PON_out, nlt_zaero_out, nlt_bgc_C_out, nlt_bgc_N_out, nlt_bgc_chl_out,␣
↪&
           nlt_bgc_DOC_out, nlt_bgc_DON_out, nlt_bgc_DIC_out, nlt_bgc_Fed_out, &
           nlt_bgc_Fep_out, nlt_bgc_Nit_out, nlt_bgc_Am_out, nlt_bgc_Sil_out, &
           nlt_bgc_DMSPp_out, nlt_bgc_DMSPd_out, nlt_bgc_DMS_out, nlt_bgc_hum_out, &
           nlt_bgc_PON_out, nt_zbgc_frac_out, nt_bgc_S_out, nlt_chl_sw_out, &
           nlt_zaero_sw_out, &
           bio_index_o_out, bio_index_out)

        integer, intent(out), optional :: &
             nt_Tsfc_out, & ! ice/snow temperature
             nt_qice_out, & ! volume-weighted ice enthalpy (in layers)
             nt_qsno_out, & ! volume-weighted snow enthalpy (in layers)
             nt_sice_out, & ! volume-weighted ice bulk salinity (CICE grid layers)
             nt_fbri_out, & ! volume fraction of ice with dynamic salt (hinS/vicen*aicen)
             nt_iage_out, & ! volume-weighted ice age
             nt_FY_out, & ! area-weighted first-year ice area
             nt_alvl_out, & ! level ice area fraction
             nt_vlvl_out, & ! level ice volume fraction
             nt_apnd_out, & ! melt pond area fraction
             nt_hpnd_out, & ! melt pond depth
             nt_ipnd_out, & ! melt pond refrozen lid thickness
             nt_smice_out,& ! mass of ice in snow
             nt_smliq_out,& ! mass of liquid water in snow
             nt_rhos_out, & ! snow density
             nt_rsnw_out, & ! snow grain radius
             nt_fsd_out,  & ! floe size distribution
             nt_isosno_out,  & ! starting index for isotopes in snow
             nt_isoice_out,  & ! starting index for isotopes in ice
             nt_aero_out,    & ! starting index for aerosols in ice
```

(continues on next page)

(continued from previous page)

```
             nt_bgc_Nit_out, & ! nutrients
             nt_bgc_Am_out,  & !
             nt_bgc_Sil_out, & !
             nt_bgc_DMSPp_out,&! trace gases (skeletal layer)
             nt_bgc_DMSPd_out,&!
             nt_bgc_DMS_out, & !
             nt_bgc_hum_out, & !
             nt_bgc_PON_out, & ! zooplankton and detritus
             nlt_bgc_Nit_out,& ! nutrients
             nlt_bgc_Am_out, & !
             nlt_bgc_Sil_out,& !
             nlt_bgc_DMSPp_out,&! trace gases (skeletal layer)
             nlt_bgc_DMSPd_out,&!
             nlt_bgc_DMS_out,& !
             nlt_bgc_hum_out,& !
             nlt_bgc_PON_out,& ! zooplankton and detritus
             nt_zbgc_frac_out,&! fraction of tracer in the mobile phase
             nt_bgc_S_out,   & ! (deprecated, was related to zsalinity)
             nlt_chl_sw_out    ! points to total chla in trcrn_sw

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             bio_index_o_out, &
             bio_index_out

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_bgc_N_out ,  & ! diatoms, phaeocystis, pico/small
             nt_bgc_C_out ,  & ! diatoms, phaeocystis, pico/small
             nt_bgc_chl_out, & ! diatoms, phaeocystis, pico/small
             nlt_bgc_N_out , & ! diatoms, phaeocystis, pico/small
             nlt_bgc_C_out , & ! diatoms, phaeocystis, pico/small
             nlt_bgc_chl_out   ! diatoms, phaeocystis, pico/small

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_bgc_DOC_out, & !  dissolved organic carbon
             nlt_bgc_DOC_out   !  dissolved organic carbon

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_bgc_DON_out, & !  dissolved organic nitrogen
             nlt_bgc_DON_out   !  dissolved organic nitrogen

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_bgc_DIC_out, & ! dissolved inorganic carbon
             nlt_bgc_DIC_out   !  dissolved inorganic carbon

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_bgc_Fed_out, & !  dissolved iron
             nt_bgc_Fep_out, & !  particulate iron
             nlt_bgc_Fed_out,& !  dissolved iron
             nlt_bgc_Fep_out   !  particulate iron

        integer (kind=int_kind), dimension(:), intent(out), optional :: &
             nt_zaero_out,   & !  black carbon and other aerosols
```

(continues on next page)

(continued from previous page)

```
             nlt_zaero_out,  & !  black carbon and other aerosols
             nlt_zaero_sw_out  ! black carbon and dust in trcrn_sw
```

icepack_write_tracer_indices

```
! write the number of column tracer indices

      subroutine icepack_write_tracer_indices(iounit)

        integer, intent(in) :: iounit
```

icepack_init_tracer_sizes

```
! set the number of column tracers

      subroutine icepack_init_tracer_sizes(&
         ncat_in, nilyr_in, nslyr_in, nblyr_in, nfsd_in  , &
         n_algae_in, n_DOC_in, n_aero_in, n_iso_in, &
         n_DON_in, n_DIC_in, n_fed_in, n_fep_in, n_zaero_in, &
         ntrcr_in, ntrcr_o_in, nbtrcr_in, nbtrcr_sw_in)

      integer (kind=int_kind), intent(in), optional :: &
         ncat_in   , & ! Categories
         nfsd_in   , & !
         nilyr_in  , & ! Layers
         nslyr_in  , & !
         nblyr_in  , & !
         n_algae_in, & ! Dimensions
         n_DOC_in  , & !
         n_DON_in  , & !
         n_DIC_in  , & !
         n_fed_in  , & !
         n_fep_in  , & !
         n_zaero_in, & !
         n_iso_in  , & !
         n_aero_in , & !
         ntrcr_in  , & ! number of tracers in use
         ntrcr_o_in, & ! number of non-bio tracers in use
         nbtrcr_in , & ! number of bio tracers in use
         nbtrcr_sw_in  ! number of shortwave bio tracers in use
```

icepack_query_tracer_sizes

```
! query the number of column tracers

      subroutine icepack_query_tracer_sizes(&
         max_algae_out  , max_dic_out    , max_doc_out      , &
         max_don_out    , max_fe_out     , nmodal1_out      , &
         nmodal2_out    , max_aero_out   , max_nbtrcr_out   , &
         ncat_out, nilyr_out, nslyr_out, nblyr_out, nfsd_out, &
         n_algae_out, n_DOC_out, n_aero_out, n_iso_out, &
         n_DON_out, n_DIC_out, n_fed_out, n_fep_out, n_zaero_out, &
         ntrcr_out, ntrcr_o_out, nbtrcr_out, nbtrcr_sw_out)

      integer (kind=int_kind), intent(out), optional :: &
         max_algae_out  , & ! maximum number of algal types
         max_dic_out    , & ! maximum number of dissolved inorganic carbon types
         max_doc_out    , & ! maximum number of dissolved organic carbon types
         max_don_out    , & ! maximum number of dissolved organic nitrogen types
         max_fe_out     , & ! maximum number of iron types
         nmodal1_out    , & ! dimension for modal aerosol radiation parameters
         nmodal2_out    , & ! dimension for modal aerosol radiation parameters
         max_aero_out   , & ! maximum number of aerosols
         max_nbtrcr_out     ! algal nitrogen and chlorophyll

      integer (kind=int_kind), intent(out), optional :: &
         ncat_out   , & ! Categories
         nfsd_out   , & !
         nilyr_out  , & ! Layers
         nslyr_out  , & !
         nblyr_out  , & !
         n_algae_out, & ! Dimensions
         n_DOC_out  , & !
         n_DON_out  , & !
         n_DIC_out  , & !
         n_fed_out  , & !
         n_fep_out  , & !
         n_zaero_out, & !
         n_iso_out  , & !
         n_aero_out , & !
         ntrcr_out  , & ! number of tracers in use
         ntrcr_o_out, & ! number of non-bio tracers in use
         nbtrcr_out , & ! number of bio tracers in use
         nbtrcr_sw_out  ! number of shortwave bio tracers in use
```

icepack_write_tracer_sizes

```
! write the number of column tracers

      subroutine icepack_write_tracer_sizes(iounit)

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

icepack_compute_tracers

```
! Compute tracer fields.
! Given atrcrn = aicen*trcrn (or vicen*trcrn, vsnon*trcrn), compute trcrn.

      subroutine icepack_compute_tracers (ntrcr,     trcr_depend,    &
                                          atrcrn,    aicen,          &
                                          vicen,     vsnon,          &
                                          trcr_base, n_trcr_strata,  &
                                          nt_strata, trcrn, Tf)

      integer (kind=int_kind), intent(in) :: &
         ntrcr                 ! number of tracers in use

      integer (kind=int_kind), dimension (ntrcr), intent(in) :: &
         trcr_depend, & ! = 0 for aicen tracers, 1 for vicen, 2 for vsnon
         n_trcr_strata  ! number of underlying tracer layers

      real (kind=dbl_kind), dimension (:,:), intent(in) :: &
         trcr_base      ! = 0 or 1 depending on tracer dependency
                        ! argument 2:  (1) aice, (2) vice, (3) vsno

      integer (kind=int_kind), dimension (:,:), intent(in) :: &
         nt_strata      ! indices of underlying tracer layers

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         atrcrn    ! aicen*trcrn or vicen*trcrn or vsnon*trcrn

      real (kind=dbl_kind), intent(in) :: &
         aicen , & ! concentration of ice
         vicen , & ! volume per unit area of ice          (m)
         vsnon     ! volume per unit area of snow         (m)

      real (kind=dbl_kind), dimension (ntrcr), intent(out) :: &
         trcrn     ! ice tracers

      real (kind=dbl_kind), intent(in) :: &
         Tf        ! Freezing point
```

4.4.17 icepack_warnings.F90

icepack_warnings_aborted

```
! turn on the abort flag in the icepack warnings package
! pass in an optional error message

      logical function icepack_warnings_aborted(instring)

        character(len=*),intent(in), optional :: instring
```

icepack_warnings_clear

```
! clear all warning messages from the icepack warning buffer

      subroutine icepack_warnings_clear()
```

icepack_warnings_clear

```
! return an array of all the current warning messages

      subroutine icepack_warnings_getall(warningsOut)

        character(len=char_len_long), dimension(:), allocatable, intent(out) :: &
             warningsOut
```

icepack_warnings_print

```
! print all warning messages from the icepack warning buffer

      subroutine icepack_warnings_print(iounit)

        integer, intent(in) :: iounit
```

icepack_warnings_flush

```
! print and clear all warning messages from the icepack warning buffer

      subroutine icepack_warnings_flush(iounit)

        integer, intent(in) :: iounit
```

4.4.18 icepack_wavefracspec.F90

icepack_init_wave

```
!  Initialize the wave spectrum and frequencies for the FSD
!
!  authors: 2018 Lettie Roach, NIWA/VUW

      subroutine icepack_init_wave(nfreq,                 &
                                   wave_spectrum_profile, &
                                   wavefreq, dwavefreq)

      integer(kind=int_kind), intent(in) :: &
         nfreq                    ! number of wave frequencies

      real(kind=dbl_kind), dimension(:), intent(out) :: &
         wave_spectrum_profile, & ! ocean surface wave spectrum as a function of␣
↪frequency
                                  ! power spectral density of surface elevation, E(f)␣
↪(units m^2 s)
         wavefreq,              & ! wave frequencies (s^-1)
         dwavefreq                ! wave frequency bin widths (s^-1)
```

icepack_step_wavefracture

```
!
!  Given fracture histogram computed from local wave spectrum, evolve
!  the floe size distribution
!
!  authors: 2018 Lettie Roach, NIWA/VUW
!
      subroutine icepack_step_wavefracture(wave_spec_type,   &
                  dt,            ncat,            nfsd,      &
                  nfreq,                                     &
                  aice,          vice,            aicen,     &
                  floe_rad_l,    floe_rad_c,                 &
                  wave_spectrum, wavefreq,        dwavefreq, &
                  trcrn,         d_afsd_wave)


      character (len=char_len), intent(in) :: &
         wave_spec_type   ! type of wave spectrum forcing

      integer (kind=int_kind), intent(in) :: &
         nfreq,        & ! number of wave frequency categories
         ncat,         & ! number of thickness categories
         nfsd            ! number of floe size categories

      real (kind=dbl_kind), intent(in) :: &
         dt,           & ! time step
         aice,         & ! ice area fraction
```

(continues on next page)

(continued from previous page)

```
         vice             ! ice volume per unit area

      real (kind=dbl_kind), dimension(ncat), intent(in) :: &
         aicen            ! ice area fraction (categories)

      real(kind=dbl_kind), dimension(:), intent(in) ::  &
         floe_rad_l,    & ! fsd size lower bound in m (radius)
         floe_rad_c       ! fsd size bin centre in m (radius)

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         wavefreq,      & ! wave frequencies (s^-1)
         dwavefreq        ! wave frequency bin widths (s^-1)

      real (kind=dbl_kind), dimension(:), intent(in) :: &
         wave_spectrum    ! ocean surface wave spectrum as a function of frequency
                          ! power spectral density of surface elevation, E(f) (units m^2␣
↪s)

      real (kind=dbl_kind), dimension(:,:), intent(inout) :: &
         trcrn            ! tracer array

      real (kind=dbl_kind), dimension(:), intent(out) :: &
         d_afsd_wave      ! change in fsd due to waves

      real (kind=dbl_kind), dimension(nfsd,ncat) :: &
         d_afsdn_wave     ! change in fsd due to waves, per category
```

4.4.19 icepack_zbgc.F90

icepack_init_bgc

```
!
      subroutine icepack_init_bgc(ncat, nblyr, nilyr, ntrcr_o, &
         cgrid, igrid, ntrcr, nbtrcr, &
         sicen, trcrn, sss, ocean_bio_all)

      integer (kind=int_kind), intent(in) :: &
         ncat   , & ! number of thickness categories
         nilyr  , & ! number of ice layers
         nblyr  , & ! number of bio layers
         ntrcr_o,& ! number of tracers not including bgc
         ntrcr  , & ! number of tracers in use
         nbtrcr     ! number of bio tracers in use

      real (kind=dbl_kind), dimension (nblyr+1), intent(inout) :: &
         igrid      ! biology vertical interface points

      real (kind=dbl_kind), dimension (nilyr+1), intent(inout) :: &
         cgrid      ! CICE vertical coordinate

```

(continues on next page)

(continued from previous page)

```
      real (kind=dbl_kind), dimension(nilyr, ncat), intent(in) :: &
         sicen     ! salinity on the cice grid

      real (kind=dbl_kind), dimension (:,:), intent(inout) :: &
         trcrn     ! subset of tracer array (only bgc)

      real (kind=dbl_kind), intent(in) :: &
         sss       ! sea surface salinity (ppt)

      real (kind=dbl_kind), dimension (:), intent(inout) :: &
         ocean_bio_all   ! fixed order, all values even for tracers false
```

icepack_init_zbgc

```
!

      subroutine icepack_init_zbgc ( &
                 R_Si2N_in, R_S2N_in, R_Fe2C_in, R_Fe2N_in, R_C2N_in, R_C2N_DON_in, &
                 R_chl2N_in, F_abs_chl_in, R_Fe2DON_in, R_Fe2DOC_in, chlabs_in, &
                 alpha2max_low_in, beta2max_in, mu_max_in, fr_graze_in, mort_pre_in, &
                 mort_Tdep_in, k_exude_in, K_Nit_in, K_Am_in, K_sil_in, K_Fe_in, &
                 f_don_in, kn_bac_in, f_don_Am_in, f_doc_in, f_exude_in, k_bac_in, &
                 grow_Tdep_in, zbgc_frac_init_in, &
                 zbgc_init_frac_in, tau_ret_in, tau_rel_in, bgc_tracer_type_in, &
                 fr_resp_in, algal_vel_in, R_dFe2dust_in, dustFe_sol_in, T_max_in, &
                 op_dep_min_in, fr_graze_s_in, fr_graze_e_in, fr_mort2min_in, fr_dFe_in,␣
↪&
                 k_nitrif_in, t_iron_conv_in, max_loss_in, max_dfe_doc1_in, &
                 fr_resp_s_in, y_sk_DMS_in, t_sk_conv_in, t_sk_ox_in, fsal_in)

      real (kind=dbl_kind), optional :: R_C2N_in(:)        ! algal C to N (mole/mole)
      real (kind=dbl_kind), optional :: R_chl2N_in(:)      ! 3 algal chlorophyll to N␣
↪(mg/mmol)
      real (kind=dbl_kind), optional :: F_abs_chl_in(:)    ! to scale absorption in Dedd
      real (kind=dbl_kind), optional :: R_C2N_DON_in(:)    ! increase compare to algal R_
↪Fe2C
      real (kind=dbl_kind), optional :: R_Si2N_in(:)       ! algal Sil to N (mole/mole)
      real (kind=dbl_kind), optional :: R_S2N_in(:)        ! algal S to N (mole/mole)
      real (kind=dbl_kind), optional :: R_Fe2C_in(:)       ! algal Fe to carbon (umol/
↪mmol)
      real (kind=dbl_kind), optional :: R_Fe2N_in(:)       ! algal Fe to N (umol/mmol)
      real (kind=dbl_kind), optional :: R_Fe2DON_in(:)     ! Fe to N of DON (nmol/umol)
      real (kind=dbl_kind), optional :: R_Fe2DOC_in(:)     ! Fe to C of DOC (nmol/umol)

      real (kind=dbl_kind), optional :: fr_resp_in         ! frac of algal growth lost␣
↪due to respiration
      real (kind=dbl_kind), optional :: algal_vel_in       ! 0.5 cm/d(m/s) Lavoie 2005 ␣
↪1.5 cm/day
      real (kind=dbl_kind), optional :: R_dFe2dust_in      ! g/g (3.5% content)␣
↪Tagliabue 2009
```

(continues on next page)

(continued from previous page)

```
      real (kind=dbl_kind), optional :: dustFe_sol_in      ! solubility fraction
      real (kind=dbl_kind), optional :: T_max_in           ! maximum temperature (C)
      real (kind=dbl_kind), optional :: op_dep_min_in      ! Light attenuates for␣
↪optical depths exceeding min
      real (kind=dbl_kind), optional :: fr_graze_s_in      ! fraction of grazing spilled␣
↪or slopped
      real (kind=dbl_kind), optional :: fr_graze_e_in      ! fraction of assimilation␣
↪excreted
      real (kind=dbl_kind), optional :: fr_mort2min_in     ! fractionation of mortality␣
↪to Am
      real (kind=dbl_kind), optional :: fr_dFe_in          ! fraction of remineralized␣
↪nitrogen
                                                           ! (in units of algal iron)
      real (kind=dbl_kind), optional :: k_nitrif_in        ! nitrification rate (1/day)
      real (kind=dbl_kind), optional :: t_iron_conv_in     ! desorption loss pFe to dFe␣
↪(day)
      real (kind=dbl_kind), optional :: max_loss_in        ! restrict uptake to % of␣
↪remaining value
      real (kind=dbl_kind), optional :: max_dfe_doc1_in    ! max ratio of dFe to␣
↪saccharides in the ice (nM Fe/muM C)
      real (kind=dbl_kind), optional :: fr_resp_s_in       ! DMSPd fraction of␣
↪respiration loss as DMSPd
      real (kind=dbl_kind), optional :: y_sk_DMS_in        ! fraction conversion given␣
↪high yield
      real (kind=dbl_kind), optional :: t_sk_conv_in       ! Stefels conversion time (d)
      real (kind=dbl_kind), optional :: t_sk_ox_in         ! DMS oxidation time (d)
      real (kind=dbl_kind), optional :: fsal_in            ! salinity limitation factor␣
↪(1)

      real (kind=dbl_kind), optional :: chlabs_in(:)       ! chla absorption 1/m/(mg/m^3)
      real (kind=dbl_kind), optional :: alpha2max_low_in(:)  ! light limitation (1/(W/m^
↪2))
      real (kind=dbl_kind), optional :: beta2max_in(:)     ! light inhibition (1/(W/m^2))
      real (kind=dbl_kind), optional :: mu_max_in(:)       ! maximum growth rate (1/d)
      real (kind=dbl_kind), optional :: grow_Tdep_in(:)    ! T dependence of growth (1/C)
      real (kind=dbl_kind), optional :: fr_graze_in(:)     ! fraction of algae grazed
      real (kind=dbl_kind), optional :: mort_pre_in(:)     ! mortality (1/day)
      real (kind=dbl_kind), optional :: mort_Tdep_in(:)    ! T dependence of mortality␣
↪(1/C)
      real (kind=dbl_kind), optional :: k_exude_in(:)      ! algal carbon  exudation␣
↪rate (1/d)
      real (kind=dbl_kind), optional :: K_Nit_in(:)        ! nitrate half saturation␣
↪(mmol/m^3)
      real (kind=dbl_kind), optional :: K_Am_in(:)         ! ammonium half saturation␣
↪(mmol/m^3)
      real (kind=dbl_kind), optional :: K_Sil_in(:)        ! silicon half saturation␣
↪(mmol/m^3)
      real (kind=dbl_kind), optional :: K_Fe_in(:)         ! iron half saturation  or␣
↪micromol/m^3
      real (kind=dbl_kind), optional :: f_don_in(:)        ! fraction of spilled grazing␣
↪to DON
      real (kind=dbl_kind), optional :: kn_bac_in(:)       ! Bacterial degredation of␣
```

(continues on next page)

(continued from previous page)

```
↪DON (1/d)
      real (kind=dbl_kind), optional :: f_don_Am_in(:)     ! fraction of remineralized␣
↪DON to Am
      real (kind=dbl_kind), optional :: f_doc_in(:)        ! fraction of mort_N that␣
↪goes to each doc pool
      real (kind=dbl_kind), optional :: f_exude_in(:)      ! fraction of exuded carbon␣
↪to each DOC pool
      real (kind=dbl_kind), optional :: k_bac_in(:)        ! Bacterial degredation of␣
↪DOC (1/d)

      real (kind=dbl_kind), optional :: zbgc_frac_init_in(:)  ! initializes mobile␣
↪fraction
      real (kind=dbl_kind), optional :: bgc_tracer_type_in(:) ! described tracer in␣
↪mobile or stationary phases
      real (kind=dbl_kind), optional :: zbgc_init_frac_in(:)  ! fraction of ocean tracer␣
↪ concentration in new ice
      real (kind=dbl_kind), optional :: tau_ret_in(:)         ! retention timescale  (s),
↪ mobile to stationary phase
      real (kind=dbl_kind), optional :: tau_rel_in(:)         ! release timescale    (s),
↪ stationary to mobile phase
```

icepack_biogeochemistry

```
!

      subroutine icepack_biogeochemistry(dt, &
                           ntrcr, nbtrcr,  &
                           upNO, upNH, iDi, iki, zfswin, &
                           zsal_tot, darcy_V, grow_net,  &
                           PP_net, hbri,dhbr_bot, dhbr_top, Zoo,&
                           fbio_snoice, fbio_atmice, ocean_bio, &
                           first_ice, fswpenln, bphi, bTiz, ice_bio_net,   &
                           snow_bio_net, fswthrun, Rayleigh_criteria, &
                           sice_rho, fzsal, fzsal_g, &
                           bgrid, igrid, icgrid, cgrid,  &
                           nblyr, nilyr, nslyr, n_algae, n_zaero, ncat, &
                           n_doc, n_dic,  n_don, n_fed, n_fep,   &
                           meltbn, melttn, congeln, snoicen, &
                           sst, sss, fsnow, meltsn, &
                           hin_old, flux_bio, flux_bio_atm, &
                           aicen_init, vicen_init, aicen, vicen, vsnon, &
                           aice0, trcrn, vsnon_init, skl_bgc)

      real (kind=dbl_kind), intent(in) :: &
         dt      ! time step

      integer (kind=int_kind), intent(in) :: &
         ncat, &
         nilyr, &
         nslyr, &
```

(continues on next page)

(continued from previous page)

```
         nblyr, &
         ntrcr, &
         nbtrcr, &
         n_algae, n_zaero, &
         n_doc, n_dic,  n_don, n_fed, n_fep

      real (kind=dbl_kind), dimension (:), intent(inout) :: &
         bgrid          , &  ! biology nondimensional vertical grid points
         igrid          , &  ! biology vertical interface points
         cgrid          , &  ! CICE vertical coordinate
         icgrid         , &  ! interface grid for CICE (shortwave variable)
         ocean_bio      , &  ! contains all the ocean bgc tracer concentrations
         fbio_snoice    , &  ! fluxes from snow to ice
         fbio_atmice    , &  ! fluxes from atm to ice
         dhbr_top       , &  ! brine top change
         dhbr_bot       , &  ! brine bottom change
         darcy_V        , &  ! darcy velocity positive up (m/s)
         hin_old        , &  ! old ice thickness
         sice_rho       , &  ! avg sea ice density  (kg/m^3)
         ice_bio_net    , &  ! depth integrated tracer (mmol/m^2)
         snow_bio_net   , &  ! depth integrated snow tracer (mmol/m^2)
         flux_bio     ! all bio fluxes to ocean

      logical (kind=log_kind), dimension (:), intent(inout) :: &
         first_ice      ! distinguishes ice that disappears (e.g. melts)
                        ! and reappears (e.g. transport) in a grid cell
                        ! during a single time step from ice that was
                        ! there the entire time step (true until ice forms)

      real (kind=dbl_kind), dimension (:,:), intent(inout) :: &
         Zoo             , & ! N losses accumulated in timestep (ie. zooplankton/bacteria)
                             ! mmol/m^3
         bphi            , & ! porosity of layers
         bTiz            , & ! layer temperatures interpolated on bio grid (C)
         zfswin          , & ! Shortwave flux into layers interpolated on bio grid  (W/m^
↪2)
         iDi             , & ! igrid Diffusivity (m^2/s)
         iki             , & ! Ice permeability (m^2)
         trcrn     ! tracers

      real (kind=dbl_kind), intent(inout) :: &
         grow_net        , & ! Specific growth rate (/s) per grid cell
         PP_net          , & ! Total production (mg C/m^2/s) per grid cell
         hbri            , & ! brine height, area-averaged for comparison with hi (m)
         upNO            , & ! nitrate uptake rate (mmol/m^2/d) times aice
         upNH                ! ammonium uptake rate (mmol/m^2/d) times aice

      real (kind=dbl_kind), intent(inout), optional :: &
         zsal_tot            ! Total ice salinity in per grid cell (g/m^2) (deprecated)

      real (kind=dbl_kind), intent(inout), optional :: &
         fzsal           , & ! Total flux  of salt to ocean at time step for conservation␣
```

(continues on next page)

(continued from previous page)

```
↪(deprecated)
        fzsal_g            ! Total gravity drainage flux (deprecated)

      logical (kind=log_kind), intent(inout), optional :: &
         Rayleigh_criteria    ! .true. means Ra_c was reached (deprecated)

      real (kind=dbl_kind), dimension (:,:), intent(in) :: &
         fswpenln        ! visible SW entering ice layers (W m-2)

      real (kind=dbl_kind), dimension (:), intent(in) :: &
         fswthrun    , & ! SW through ice to ocean            (W/m^2)
         meltsn      , & ! snow melt in category n (m)
         melttn      , & ! top melt in category n (m)
         meltbn      , & ! bottom melt in category n (m)
         congeln     , & ! congelation ice formation in category n (m)
         snoicen     , & ! snow-ice formation in category n (m)
         flux_bio_atm, & ! all bio fluxes to ice from atmosphere
         aicen_init  , & ! initial ice concentration, for linear ITD
         vicen_init  , & ! initial ice volume (m), for linear ITD
         vsnon_init  , & ! initial snow volume (m), for aerosol
         aicen , & ! concentration of ice
         vicen , & ! volume per unit area of ice          (m)
         vsnon     ! volume per unit area of snow         (m)

      real (kind=dbl_kind), intent(in) :: &
         aice0   , & ! open water area fraction
         sss     , & ! sea surface salinity (ppt)
         sst     , & ! sea surface temperature (C)
         fsnow       ! snowfall rate (kg/m^2 s)

      logical (kind=log_kind), intent(in) :: &
         skl_bgc       ! if true, solve skeletal biochemistry
```

icepack_load_ocean_bio_array

```
! basic initialization for ocean_bio_all

      subroutine icepack_load_ocean_bio_array(max_nbtrcr, &
          max_algae, max_don, max_doc, max_dic, max_aero, max_fe, &
          nit, amm, sil, dmsp, dms, algalN, &
          doc, don, dic, fed, fep, zaeros, ocean_bio_all, hum)

      integer (kind=int_kind), intent(in) :: &
         max_algae   , & ! maximum number of algal types
         max_dic     , & ! maximum number of dissolved inorganic carbon types
         max_doc     , & ! maximum number of dissolved organic carbon types
         max_don     , & ! maximum number of dissolved organic nitrogen types
         max_fe      , & ! maximum number of iron types
         max_aero    , & ! maximum number of aerosols
         max_nbtrcr      ! maximum number of bio tracers
```

(continues on next page)

(continued from previous page)

```
      real (kind=dbl_kind), intent(in) :: &
         nit           , & ! ocean nitrate (mmol/m^3)
         amm           , & ! ammonia/um (mmol/m^3)
         sil           , & ! silicate (mmol/m^3)
         dmsp          , & ! dmsp (mmol/m^3)
         dms           , & ! dms (mmol/m^3)
         hum               ! humic material (mmol/m^3)

      real (kind=dbl_kind), dimension (max_algae), intent(in) :: &
         algalN            ! ocean algal nitrogen (mmol/m^3) (diatoms, phaeo, pico)

      real (kind=dbl_kind), dimension (max_doc), intent(in) :: &
         doc               ! ocean doc (mmol/m^3)  (proteins, EPS, lipid)

      real (kind=dbl_kind), dimension (max_don), intent(in) :: &
         don               ! ocean don (mmol/m^3)

      real (kind=dbl_kind), dimension (max_dic), intent(in) :: &
         dic               ! ocean dic (mmol/m^3)

      real (kind=dbl_kind), dimension (max_fe), intent(in) :: &
         fed, fep          ! ocean disolved and particulate fe (nM)

      real (kind=dbl_kind), dimension (max_aero), intent(in) :: &
         zaeros            ! ocean aerosols (mmol/m^3)

      real (kind=dbl_kind), dimension (max_nbtrcr), intent(inout) :: &
         ocean_bio_all     ! fixed order, all values even for tracers false
```

icepack_init_ocean_bio

```
!  Initialize ocean concentration

      subroutine icepack_init_ocean_bio (amm, dmsp, dms, algalN, doc, dic, don, &
             fed, fep, hum, nit, sil, zaeros, max_dic, max_don, max_fe, max_aero,&
             CToN, CToN_DON)

      integer (kind=int_kind), intent(in) :: &
        max_dic, &
        max_don, &
        max_fe, &
        max_aero

      real (kind=dbl_kind), intent(out):: &
       amm      , & ! ammonium
       dmsp     , & ! DMSPp
       dms      , & ! DMS
       hum      , & ! humic material
       nit      , & ! nitrate
```

(continues on next page)

(continued from previous page)

```
       sil          ! silicate

      real (kind=dbl_kind), dimension(:), intent(out):: &
       algalN    , & ! algae
       doc       , & ! DOC
       dic       , & ! DIC
       don       , & ! DON
       fed       , & ! Dissolved Iron
       fep       , & ! Particulate Iron
       zaeros        ! BC and dust

      real (kind=dbl_kind), dimension(:), intent(inout), optional :: &
       CToN      , & ! carbon to nitrogen ratio for algae
       CToN_DON      ! nitrogen to carbon ratio for proteins
```

CHAPTER FIVE

DEVELOPER GUIDE

5.1 About Development

The Icepack model consists of three different parts, the column physics code, the icepack driver, and the scripts. Development of each of these pieces will be described below separately.

Subroutine calls and other linkages into Icepack from the host model should only need to access the **icepack_intfc*.F90** interface modules within the `columnphysics/` directory. The Icepack driver in the `configuration/driver/` directory is based on the CICE model and provides an example of the sea ice host model capabilities needed for inclusion of Icepack. In particular, host models will need to include code equivalent to that in the modules **icedrv_*_column.F90**. Calls into the Icepack interface routines are primarily from **icedrv_step_mod.F90** but there are others (search the driver code for `intfc`).

Guiding principles for the creation of Icepack include the following:

- The column physics modules shall be independent of all sea ice model infrastructural elements that may vary from model to model. Examples include input/output, timers, references to CPUs or computational tasks, initialization other than that necessary for strictly physical reasons, and anything related to a horizontal grid.
- The column physics modules shall not call or reference any routines or code that reside outside of the **columnphysics/** directory.
- Any capabilities required by a host sea ice model (e.g. calendar variables, tracer flags, diagnostics) shall be implemented in the driver and passed into or out of the column physics modules via array arguments.

5.1.1 Git workflow and Pull Requests

There is extensive Information for Developers documentation available. See https://github.com/CICE-Consortium/About-Us/wiki/Resource-Index#information-for-developers for information on:

- Contributing to model development
- Software development practices guide
- git Workflow Guide - including extensive information about the Pull Request process and requirements
- Documentation Workflow Guide

5.2 Icepack Column Physics

5.2.1 File List

The column physics source code contains the following files

columnphysics/ the column physics code

- **icepack_aerosol.F90** handles most work associated with the aerosol tracers
- **icepack_age.F90** handles most work associated with the age tracer
- **icepack_algae.F90** biogeochemistry
- **icepack_atmo.F90** stability-based parameterization for calculation of turbulent ice–atmosphere fluxes
- **icepack_brine.F90** evolves the brine height tracer
- **icepack_firstyear.F90** handles most work associated with the first-year ice area tracer
- **icepack_flux.F90** fluxes needed/produced by the model
- **icepack_fsd.F90** supports floe size distribution
- **icepack_intfc.F90** interface routines for linking Icepack with a host sea ice model
- **icepack_isotope.F90** handles isotopes
- **icepack_itd.F90** utilities for managing ice thickness distribution
- **icepack_kinds.F90** basic definitions of reals, integers, etc.
- **icepack_mechred.F90** mechanical redistribution (ridging)
- **icepack_meltpond_lvl.F90** level-ice melt pond parameterization
- **icepack_meltpond_topo.F90** topo melt pond parameterization
- **icepack_mushy_physics.F90** physics routines for mushy thermodynamics
- **icepack_ocean.F90** mixed layer ocean model
- **icepack_orbital.F90** orbital parameters for Delta-Eddington shortwave parameterization
- **icepack_parameters.F90** basic model parameters including physical and numerical constants requried for column package
- **icepack_shortwave.F90** shortwave and albedo parameterizations
- **icepack_snow.F90** snow physics
- **icepack_therm_bl99.F90** multilayer thermodynamics of [6]
- **icepack_therm_itd.F90** thermodynamic changes mostly related to ice thickness distribution
- **icepack_therm_mushy.F90** mushy-theory thermodynamics of [71]
- **icepack_therm_shared.F90** code shared by all thermodynamics parameterizations
- **icepack_therm_vertical.F90** vertical growth rates and fluxes
- **icepack_tracers.F90** tracer information
- **icepack_warnings.F90** utilities for writing warning and error messages
- **icepack_wavefracspec.F90** wave impact on sea ice
- **icepack_zbgc.F90** driver for ice biogeochemistry and brine tracer motion
- **icepack_zbgc_shared.F90** parameters and shared code for biogeochemistry and brine height

5.2.2 Coding Standard

The column physics is a library that solves the sea ice column physics on a timestep by timestep and gridpoint by gridpoint basis. It consists of Fortran routines with input and output arguments. The model state is saved in the host model. There is no communication between gridcells so the underlying implementation supports no parallelization. It however can be called in parallel by a driver that is running on multiple pes with a decomposed grid.

The column physics does not have a time manager. Calendaring is expected to be dealt with by the host model. The column physics does not read any forcing data, that is passed into the column physics though interfaces. In fact, there are no direct IO capabilities in the column physics. That is to say, the column physics does not open files to read or write. However, the column physics contains a warning and abort package (see section *Error Messages and Aborts*) that provides the column package with the ability to store log output. That output can be queried by the host model or it can be written directly via a specific routine. The warning package also provides access to an abort flag that the host model can query after each call to check for successful completion of the column physics package.

All column physics public interfaces and public data are defined in the **icepack_intfc.F90** file (see section *Access*). Internal column physics settings should all be accessible through interfaces. The internal constants, parameters, and tracer settings have init (set), query (get), and write interfaces that provides access to internal column physics settings. The host model should not have to use "use" statements to access any of the column physics data outside of what is provided through the icepack_intfc module.

The public column physics interfaces use optional arguments where it makes sense and there is an ongoing effort to extend the optional arguments supported. It's strongly recommended that calls to the icepack interfaces be done with keyword=value arguments. All icepack arguments support this method.

Overall, columnphysics changes in the Icepack model should include the following

- All modules should have the following set at the top

```
implicit none
private
```

- Any public module interfaces or data should be explicitly specified
- All subroutines and functions should define the subname character parameter statement to match the interface name like

```
character(len=*),parameter :: subname='(lateral_melt_bgc)'
```

- All interfaces that are public outside the Icepack columnphysics should include autodocument_start and autodocument_end comment lines with appropriate syntax and location. If any interfaces are added or updated, then the internal documentation should be updated via

```
./icepack.setup --docintfc
```

  See also *Public Interfaces* for more information about the docintfc option.

- The icepack_warnings package should be used to cache log messages and set the abort flag. To add a log message, use icepack_warnings_add like

```
call icepack_warnings_add(subname//' algorithm did not converge')
```

  To formally set the abort flag, use

```
call icepack_warnings_setabort(.true.,__FILE__,__LINE__)
```

  See also *Error Messages and Aborts* for more information about how the external calling program will write those message and check whether Icepack aborted.

- Every interface call within the columnphysics should be followed by

  ```
  if (icepack_warnings_aborted(subname)) return
  ```

  to support errors backing up the call tree to the external program
- Variables defined in icepack_kinds, icepack_tracers, icepack_parameters, and icepack_orbital should be accessed from WITHIN Icepack by Fortran use statements (even though this is not recommended for drivers). It's also possible to use the public methods to access internal Icepack variables. Again, from the icepack driver or other external programs, the columnphysics variables should ALWAYS be access thru the interface methods and icepack_intfc (see also *Access*).
- Icepack is a simple serial code. Global flags and parameters should be set identically on all tasks/threads that call into Icepack. Icepack has no ability to reconcile or identify inconsistencies between different tasks/threads. All aspects of correct parallel implementation is managed by the driver code.
- Optional arguments are encouraged in the public Icepack interfaces. They provide backwards compatibility in the public interfaces and allow future extensions. Arguments that are not always required should ultimately be made optional. There are several ways optional arguments can be passed down the calling tree in Icepack. Two options, copying into local data or copying into module data are viable. But the recommended approach is to pass optional arguments down the calling tree,
  - Optional arguments can be used as calls are made down the calling tree without regard to whether they are present or not.
  - Use universal flags and parameters to turn on/off features. Avoid having features triggered by the presence of optional arguments.
  - Verify that the optional arguments required for any feature are passed in at the top level of each Icepack interface. If not, then abort.
  - Leverage the icepack subroutine `icepack_checkoptargflags` which controls how often to check the optional arguments. The `argcheck` namelist setting controls when to do the checks, 'never', 'first', or 'always' are valid settings
  - Pass optional arguments down the calling tree within Icepack as needed. In Fortran, the present attribute is carried down the calling tree automatically, but the `optional` attribute should also be defined in lower level subroutines. This is not strictly required in cases where the subroutine is always called with the optional arguments, but it's good practice.
  - An example of how this might look is

    ```
    use icepack_parameters, only: flag_arg2, flag_arg3

    subroutine icepack_public_interface(arg1, arg2, arg3, ...)
    real (kind=dbl_kind), intent(inout) :: arg1
    real (kind=dbl_kind), optional, dimension(:), intent(inout) :: arg2
    real (kind=dbl_kind), optional, intent(inout) :: arg3

    logical, save :: first_call = .true.
    character(len=*), parameter :: subname = '(icepack_public_interface)'

    if (icepack_chkoptargflag(first_call)) then
       if (flag_arg2) then
          if (.not.present(arg2)) then
             call icepack_warnings_setabort(.true.,__FILE__,__LINE__)
             call icepack_warnings_add(subname//' flag_arg2 set but arg2 not passed
    ↪')
    ```

(continues on next page)

(continued from previous page)

```
      endif
   endif
   if (flag_arg3) then
      if (.not.present(arg3)) then
         call icepack_warnings_setabort(.true.,__FILE__,__LINE__)
         call icepack_warnings_add(subname//' flag_arg3 set but arg3 not passed
→')
      endif
   endif
   if (icepack_warnings_aborted(subname)) return
endif

...
call some_columnphysics_subroutine(arg1, arg2, arg3, ...)
...

first_call = .false.

end subroutine

!------------

subroutine some_columnphysics_subroutine(arg1, arg2, arg3, ...)

real (kind=dbl_kind), intent(inout) :: arg1
real (kind=dbl_kind), optional, dimension(:), intent(inout) :: arg2
real (kind=dbl_kind), optional, intent(inout) :: arg3

if (flag_arg2) then
   arg2(:) = ...
endif

if (flag_arg3) then
   call someother_columnphysics_subroutine(arg3)
endif

end subroutine

!------------

subroutine someother_columnphysics_subroutine(arg3)

real (kind=dbl_kind), optional, intent(inout) :: arg3

arg3 = ...

end subroutine
```

Some notes

- If optional arguments are passed but not needed, this is NOT an error. If optional argument are not passed but needed, this is an error.
- If checking and implementation are done properly, optional arguments that are not needed will never be ref-

erenced anywhere in Icepack at that timestep. Optional arguments should be matched with the appropriate flags at the first entry into Icepack as much as possible.

  – There is a unit test (optarg) in CICE to verify optional argument passing. There is also a unit test (opticep) in CICE that checks that NOT passing the optional arguments from CICE is robust.

5.3 Driver Implementation

The icepack driver is Fortran source code and exists to test the column physics in a stand-alone mode for some simple column configurations.

5.3.1 File List

The icepack driver consists of the following files

configuration/driver/ driver for testing Icepack in stand-alone mode

- **icedrv_MAIN.F90** main program
- **icedrv_InitMod.F90** routines for initializing a run
- **icedrv_RunMod.F90** main driver routines for time stepping
- **icedrv_arrays_column.F90** essential arrays to describe the state of the ice
- **icedrv_calendar.F90** keeps track of what time it is
- **icedrv_constants.F90** physical and numerical constants and parameters
- **icedrv_diagnostics.F90** miscellaneous diagnostic and debugging routines
- **icedrv_diagnostics_bgc.F90** diagnostic routines for biogeochemistry
- **icedrv_domain_size.F90** domain sizes
- **icedrv_flux.F90** fluxes needed/produced by the model
- **icedrv_forcing.F90** routines to read and interpolate forcing data for stand-alone model runs
- **icedrv_forcing_bgc.F90** routines to read and interpolate forcing data for bgc stand-alone model runs
- **icedrv_init.F90** general initialization routines
- **icedrv_init_column.F90** initialization routines specific to the column physics
- **icedrv_restart.F90** driver for reading/writing restart files
- **icedrv_restart_bgc.F90** restart routines specific to the column physics
- **icedrv_restart_shared.F90** code shared by all restart options
- **icedrv_state.F90** essential arrays to describe the state of the ice
- **icedrv_step.F90** routines for time stepping the major code components
- **icedrv_system.F90** overall system management calls

5.3.2 Overview

The icepack driver exists to test the column physics. At the present time, it is hardwired to run 4 different gridcells on one processor with the same forcing used for all gridcells. There is no MPI and no threading built into the icepack driver. There is limited IO capabilities, no history files, and no netcdf restart files. The model generally runs very quickly.

Forcing data and details on these data are available in *Forcing data*.

5.4 Scripts Implementation

The scripts are the third part of the icepack package. They support setting up cases, building, and running the icepack stand-alone model.

5.4.1 File List

The directory structure under configure/scripts is as follows.

configuration/scripts/

- **Makefile** primary makefile
- **icepack.batch.csh** creates batch scripts for particular machines
- **icepack.build** compiles the code
- **icepack.launch.csh** creates script logic that runs the executable
- **icepack.run.setup.csh** sets up the run scripts
- **icepack.run.suite.csh** sets up the test suite
- **icepack.settings** defines environment, model configuration and run settings
- **icepack.test.setup.csh** creates configurations for testing the model
- **icepack_decomp.csh** defines the grid size
- **icepack_in** namelist input data
- **machines/** machine specific files to set env and Macros
- **makdep.c** determines module dependencies
- **options/** other namelist configurations available from the icepack.setup command line
- **parse_namelist.sh** replaces namelist with command-line configuration
- **parse_namelist_from_settings.sh** replaces namelist with values from icepack.settings
- **parse_settings.sh** replaces settings with command-line configuration
- **tests/** scripts for configuring and running basic tests

5.4.2 Strategy

The icepack scripts are implemented such that everything is resolved after **icepack.setup** is called. This is done by both copying specific files into the case directory and running scripts as part of the **icepack.setup** command line to setup various files.

icepack.setup drives the case setup. It is written in csh. All supporting scripts are relatively simple csh or sh scripts.

The file **icepack.settings** specifies a set of env defaults for the case. The file **icepack_in** defines the namelist input for the icepack driver.

5.4.3 Preset Case Options

`icepack.setup -s` option allows the user to choose some predetermined icepack settings and namelist. Those options are defined in **configurations/scripts/options/** and the files are prefixed by either set_env, set_nml, or test_nml. When **icepack.setup** is executed, the appropriate files are read from **configurations/scripts/options/** and the **icepack.settings** and/or **icepack_in** files are updated in the case directory based on the values in those files.

The filename suffix determines the name of the -s option. So, for instance,

```
icepack.setup -s diag1,debug,bgcISPOL
```

will search for option files with suffixes of diag1, debug, and bgcISPOL and then apply those settings.

parse_namelist.sh, **parse_settings.sh**, and **parse_namelist_from_settings.sh** are the three scripts that modify **icepack_in** and **icepack.settings**.

To add new options, just add new files to the **configurations/scripts/options/** directory with appropriate names and syntax. The set_nml file syntax is the same as namelist syntax and the set_env files are consistent with csh setenv syntax. See other files for examples of the syntax.

5.4.4 Machines

Machine specific information is contained in **configuration/scripts/machines**. That directory contains a Macros file and an env file for each supported machine. For more information on porting to a new machine, see *Porting*.

5.4.5 Test scripts

Under **configuration/scripts/tests** are several files including the scripts to setup the various tests, such as smoke and restart tests (**test_smoke.script**, **test_restart.script**). and the files that describe which options files are needed for each test (ie. **test_smoke.files**, **test_restart.files**). A baseline test script (**baseline.script**) is also there to setup the general regression and comparison testing. That directory also contains the preset test suites (ie. **base_suite.ts**) and a file that supports post-processing on the model output (**timeseries.csh**). There is also a script **report_results.csh** that pushes results from test suites back to the CICE-Consortium test results wiki page.

To add a new test (for example newtest), several files may be needed,

- **configuration/scripts/tests/test_newtest.script** defines how to run the test. This chunk of script will be incorporated into the case test script
- **configuration/scripts/tests/test_newtest.files** list the set of options files found in **configuration/scripts/options/** needed to run this test. Those files will be copied into the test directory when the test is invoked so they are available for the **test_newtest.script** to use.
- some new files may be needed in **configuration/scripts/options/**. These could be relatively generic **set_nml** or **set_env** files, or they could be test specific files typically carrying a prefix of **test_nml**.

Generating a new test, particularly the **test_newtest.script** usually takes some iteration before it's working properly.

5.5 Adding tracers

We require that any changes made to the code be implemented in such a way that they can be "turned off" through namelist flags. In most cases, code run with such changes should be bit-for-bit identical with the unmodified code. Occasionally, non-bit-for-bit changes are necessary, e.g. associated with an unavoidable change in the order of operations. In these cases, changes should be made in stages to isolate the non-bit-for-bit changes, so that those that should be bit-for-bit can be tested separately.

Tracers added to Icepack will also require extensive modifications to the host sea ice model, including initialization on the horizontal grid, namelist flags and restart capabilities. Modifications to the Icepack driver should reflect the modifications needed in the host model but are not expected to match completely. We recommend that the logical namelist variable `tr_[tracer]` be used for all calls involving the new tracer outside of **ice_[tracer].F90**, in case other users do not want to use that tracer.

A number of optional tracers are available in the code, including ice age, first-year ice area, melt pond area and volume, brine height, aerosols, and level ice area and volume (from which ridged ice quantities are derived). Salinity, enthalpies, age, aerosols, level-ice volume, brine height and most melt pond quantities are volume-weighted tracers, while first-year area, pond area, and level-ice area are area-weighted tracers. Biogeochemistry tracers in the skeletal layer are area-weighted, and vertical biogeochemistry tracers are volume-weighted. In the absence of sources and sinks, the total mass of a volume-weighted tracer such as aerosol (kg) is conserved under transport in horizontal and thickness space (the mass in a given grid cell will change), whereas the aerosol concentration (kg/m) is unchanged following the motion, and in particular, the concentration is unchanged when there is surface or basal melting. The proper units for a volume-weighted mass tracer in the tracer array are kg/m.

In several places in the code, tracer computations must be performed on the conserved "tracer volume" rather than the tracer itself; for example, the conserved quantity is $h_{pnd}a_{pnd}a_{lvl}a_i$, not h_{pnd}. Conserved quantities are thus computed according to the tracer dependencies (weights), which are tracked using the arrays `trcr_depend` (indicates dependency on area, ice volume or snow volume), `trcr_base` (a dependency mask), `n_trcr_strata` (the number of underlying tracer layers), and `nt_strata` (indices of underlying layers). See subroutine *icepack_compute_tracers* in **icepack_tracers.F90**.

To add a tracer, follow these steps using one of the existing tracers as a pattern (e.g. age).

1. **icedrv_domain_size.F90**: increase `max_ntrcr` (can also add option to **icepack.settings** and **icepack.build**)
2. **icepack_tracers.F90**:
    - declare `nt_[tracer]` and `tr_[tracer]`
    - add flags and indices to the init, query and write subroutines, and call these routines as needed throughout the code
3. **icepack_[tracer].F90**: create physics routines
4. **icedrv_init.F90**: (some of this may be done in **icepack_[tracer].F90** instead)
    - declare `tr_[tracer]` and `nt_[tracer]` as needed
    - add logical namelist variable `tr_[tracer]`
    - initialize namelist variable
    - print namelist variable to diagnostic output file
    - increment number of tracers in use based on namelist input (`ntrcr`)
    - define tracer dependencies
5. **icedrv_step_mod.F90** (and elsewhere as needed):
    - call physics routines in **icepack_[tracer].F90**
6. **icedrv_restart.F90**:

   - define restart variables
   - call routines to read, write tracer restart data

7. **icepack_in**: add namelist variables to *tracer_nml* and *icefields_nml*. Best practice is to set the namelist values so that the new capability is turned off, and create an option file with your preferred configuration in **configuration/scripts/options**.
8. If strict conservation is necessary, add diagnostics as noted for topo ponds in Section *Melt ponds*
9. Update documentation, including **icepack_index.rst** and **ug_case_settings.rst**

5.6 Adding diagnostics

Icepack produces separate ASCII (text) log output for four cells, each with a different initial condition (full ITD, slab ice, ice free, land) designated by the variable n here. Each of the diagnostic files contains the state information for that cell. The procedure for adding diagnostic variables to the output is outlined here.

1. For non-BGC variables, edit **icedrv_diagnostics.F90**:
   - If the variable is already defined within the code, then add it to a "use" statement in the subroutine `runtime_diags`.
   - Note that if the variable is not readily accessible through a use statement, then a global variable may need to be defined. This might be in **icedrv_state.F90** or **icedrv_flux.F90** for example.
   - Additionally, if the variable is a derived quantity, then the variables needed to calculate the new quantity may need to be added to a use statement. For example, see how `hiavg` and `hsavg` are computed.
   - If the variable is a scalar, then follow the example of `aice` or `hiavg`, copying the write statement to an appropriate place in the output list, and editing as needed. The format "900" is appropriate for most scalars. The following example adds snow melt (`melts`).

   ```
   use icedrv_flux, only: melts

   write(nu_diag_out+n-1,900) 'snow melt (m)          = ',melts(n) ! snow melt
   ```

   - If the variable is an array, then you can compute the mean value (e.g. `hiavg`) or print the array values (e.g. `fiso_evap`). This may requires adding the array sizes and a counter for the loop(s). E.g. to print the category ice area, `aicen` over ncat thickness categories:

   ```
   use icedrv_domain_size, only: ncat

   use icedrv_state, only: aicen

   ! local variables

   integer (kind=int_kind) :: &
      n, nc, k

   do nc = 1,ncat
      write(nu_diag_out+n-1,901) 'Category ice area =       ',aicen(n,nc),nc !␣
   ↪category ice area
   enddo
   ```

   - If the variable is a tracer, then in addition to the variable trcr or trcrn, you will need the tracer index (e.g. `nt_Tsfc`).

   - In some cases, a new format statement might be needed.

2. For BGC variables, edit **icedrv_diagnostics_bgc.F90**:

   - If the variable is already defined within the code, then add it to a "use" statement in the subroutine `hbrine_diags` or `bgc_diags` and follow a similar procedure for state variables as above.
   - Note that the BGC needs to be activated and the particular tracer turned on.

In general, try to format the output statements to line up with the surrounding print messages. This may require a couple of tries to get it to compile and run.

5.7 Other things

5.7.1 Running with a Debugger

Availability and usage of interactive debuggers varies across machines. Contact your system administrator for additional information about what's available on your system. To run with an interactive debugger, the following general steps should be taken.

- Setup a case
- Modify the env file and Macros file to add appropriate modules and compiler/ linker flags
- Build the model
- Get interactive hardware resources as needed
- Open a csh shell
- Source the env.${machine} file
- Source cice.settings
- Change directories to the run directory
- Manually launch the executable thru the debugger

CHAPTER

SIX

APPENDICES

6.1 Index of primary variables and parameters

This index defines many (but not all) of the symbols used frequently in the ice model code. Values appearing in this list are fixed or recommended; most namelist parameters are indicated (●) with their default values. All quantities in the code are expressed in MKS units (temperatures may take either Celsius or Kelvin units). Deprecated parameters are listed at the end.

Namelist variables are partly included here, but they are fully documented in section *Tables of Namelist Options*.

Table 1: *Alphabetical Index of Icepack Variables and Parameters*

A			
a_rapid_mode	● brine channel diameter		
afsd(n)	floe size distribution (in category n)		
ahmax	● thickness above which ice albedo is constant	0.3m	
aice_init	concentration of ice at beginning of timestep		
aice0	fractional open water area		
aice(n)	total concentration of ice in grid cell (in category n)		
albedo_type	● type of albedo parameterization ('default' or 'constant')		
albice	bare ice albedo		
albicei	● near infrared ice albedo for thicker ice		
albicev	● visible ice albedo for thicker ice		
albocn	ocean albedo	0.06	
albpnd	melt pond albedo		
albsno	snow albedo		
albsnowi	● near infrared, cold snow albedo		
albsnowv	● visible, cold snow albedo		
algalN	algal nitrogen concentration	mmol/m^3	
alndf	albedo: near IR, diffuse		
alndr	albedo: near IR, direct		
alvdf	albedo: visible, diffuse		
alvdr	albedo: visible, direct		
alndf_ai	grid-box-mean value of alndf		
alndr_ai	grid-box-mean value of alndr		
alvdf_ai	grid-box-mean value of alvdf		
alvdr_ai	grid-box-mean value of alvdr		
amm	ammonia/um concentration	mmol/m^3	

continues on next page

Table 1 – continued from previous page

aparticn	participation function		
apeff_ai	grid-cell-mean effective pond fraction		
apondn	area concentration of melt ponds		
araftn	area fraction of rafted ice		
ardgn	fractional area of ridged ice		
argcheck	optional argument setting	first	
aspect_rapid_mode	• brine convection aspect ratio	1	
astar	e-folding scale for participation function	0.05	
atmiter_conv	• convergence criteria for ustar	0.0	
atm_data_format	• format of atmospheric forcing files		
atm_data_type	• type of atmospheric forcing		
atmbndy	• atmo boundary layer parameterization ('similarity','constant' or 'mixed')		
awtidf	weighting factor for near-ir, diffuse albedo	0.36218	
awtidr	weighting factor for near-ir, direct albedo	0.00182	
awtvdf	weighting factor for visible, diffuse albedo	0.63282	
awtvdr	weighting factor for visible, direct albedo	0.00318	
B			
bgc_data_type	• forcing type for biogeochemistry		
bgc_flux_type	• ice-ocean flux velocity type		
bgc_tracer_type	tracer_type for bgc tracers		
bgrid	nondimensional vertical grid points for bio grid		
bignum	a large number	10^{30}	
bphi	porosity of ice layers on bio grid		
bTiz	temperature of ice layers on bio grid		
C			
calc_dragio	• if true, calculate `dragio` from `iceruf_ocn` and `thickness_ocn_layer1`	F	
calc_strair	• if true, calculate wind stress	T	
calc_Tsfc	• if true, calculate surface temperature	T	
Cdn_atm	atmospheric drag coefficient		
Cdn_ocn	ocean drag coefficient		
Cf	• ratio of ridging work to PE change in ridging	17.	
cgrid	vertical grid points for ice grid (compare bgrid)		
char_len	length of character variable strings	80	
char_len_long	length of longer character variable strings	256	
check_step	time step on which to begin writing debugging data		
cldf	cloud fraction		
cm_to_m	cm to meters conversion	0.01	
coldice	value for constant albedo parameterization	0.70	
coldsnow	value for constant albedo parameterization	0.81	
conduct	• conductivity parameterization		
congel	basal ice growth	m	
conserv_check	if true, check conservation		
coszen	cosine of the zenith angle		
Cp	proportionality constant for potential energy	$kg/m^2/s^2$	
cp_air	specific heat of air	1005.0 J/kg/K	

continues on next page

Table 1 – continued from previous page

cp_ice	specific heat of fresh ice	2106. J/kg/K	
cp_ocn	specific heat of sea water	4218. J/kg/K	
cp_wv	specific heat of water vapor	1.81x10^3 J/kg/K	
cp063	diffuse fresnel reflectivity (above)	0.063	
cp455	diffuse fresnel reflectivity (below)	0.455	
cpl_frazil	• type of frazil ice coupling		
Cs	fraction of shear energy contributing to ridging	0.25	
Cstar	constant in Hibler ice strength formula	20.	
D			
d_afsd_[proc]	change in FSD due to processes		
daice_da	data assimilation concentration increment rate		
daidtd	ice area tendency due to dynamics/transport	1/s	
daidtt	ice area tendency due to thermodynamics	1/s	
dalb_mlt	[see **icepack_shortwave.F90**]	-0.075	
dalb_mlti	[see **icepack_shortwave.F90**]	-0.100	
dalb_mltv	[see **icepack_shortwave.F90**]	-0.150	
darcy_V	Darcy velocity used for brine height tracer		
dardg1(n)dt	rate of fractional area loss by ridging ice (category n)	1/s	
dardg2(n)dt	rate of fractional area gain by new ridges (category n)	1/s	
daymo	number of days in one month		
daycal	day number at end of month		
days_per_year	• number of days in one year	365	
dbl_kind	definition of double precision	selected_real_kind(13)	
dbug	• write extra diagnostics	.false.	
depressT	ratio of freezing temperature to salinity of brine	0.054 deg/ppt	
dhbr_bt	change in brine height at the bottom of the column		
dhbr_top	change in brine height at the top of the column		
dhsn	depth difference for snow on sea ice and pond ice		
diag_file	• diagnostic output file (alternative to standard out)		
diagfreq	• how often diagnostic output is written (10 = once per 10 dt)		
divu	strain rate I component, velocity divergence	1/s	
divu_adv	divergence associated with advection	1/s	
dms	dimethyl sulfide concentration	mmol/m^3	
dmsp	dimethyl sulfoniopropionate concentration	mmol/m^3	
dpscale	• scaling factor for flushing in permeable ice (ktherm=1)	1×10^{-3}	
dragio	drag coefficient for water on ice	0.00536	
dSdt_slow_mode	• drainage strength parameter		
dsnow	change in snow thickness	m	
dt	• thermodynamics time step	3600. s	
dt_dyn	dynamics/ridging/transport time step		

continues on next page

Table 1 – continued from previous page

dT_mlt	• Δ temperature per Δ snow grain radius	1. deg	
dumpfreq	• dump frequency for restarts, y, m or d		
dumpfreq_n	• restart output frequency		
dvidtd	ice volume tendency due to dynamics/transport	m/s	
dvidtt	ice volume tendency due to thermodynamics	m/s	
dvirdg(n)dt	ice volume ridging rate (category n)	m/s	
dwavefreq	widths of wave freqency bins	1/s	
E			
eice(n)	energy of melting of ice per unit area (in category n)	J/m^2	
emissivity	emissivity of snow and ice	0.985	
eps13	a small number	10^{-13}	
eps16	a small number	10^{-16}	
esno(n)	energy of melting of snow per unit area (in category n)	J/m^2	
evap	evaporative water flux	kg/m^2/s	
F			
faero_atm	aerosol deposition rate	kg/m^2/s	
faero_ocn	aerosol flux to the ocean	kg/m^2/s	
fiso_atm	water isotope deposition rate	kg/m^2/s	
fiso_ocn	water isotope flux to the ocean	kg/m^2/s	
fiso_evap	water isotope evaporation rate	kg/m^2/s	
fbot_xfer_type	• type of heat transfer coefficient under ice		
fcondtop(n)(_f)	conductive heat flux	W/m^2	
ferrmax	max allowed energy flux error (thermodynamics)	1x 10^{-3} W/m^2	
ffracn	fraction of fsurfn used to melt pond ice		
fhocn	net heat flux to ocean	W/m^2	
fhocn_ai	grid-box-mean net heat flux to ocean (fhocn)	W/m^2	
first_ice	flag for initial ice formation		
flat	latent heat flux	W/m^2	
floe_rad_l	lower bounds for FSD size bins (radius)	m	
floe_rad_c	centers of FSD size bins (radius)	m	
floe_binwidth	width of FSD size bins (radius)	m	
floediam	effective floe diameter for lateral melt	300. m	
floeshape	floe shape constant for lateral melt	0.66	
flux_bio	all biogeochemistry fluxes passed to ocean		
flux_bio_ai	all biogeochemistry fluxes passed to ocean, grid cell mean		
flw	incoming longwave radiation	W/m^2	
flwout	outgoing longwave radiation	W/m^2	
formdrag	• calculate form drag		
fpond	fresh water flux to ponds	kg/m^2/s	
fr_resp	bgc respiration fraction	0.05	
frain	rainfall rate	kg/m^2/s	
frazil	frazil ice growth	m	
fresh	fresh water flux to ocean	kg/m^2/s	
fresh_ai	grid-box-mean fresh water flux (fresh)	kg/m^2/s	

continues on next page

Table 1 – continued from previous page

frz_onset	day of year that freezing begins		
frzmlt	freezing/melting potential	W/m^2	
frzmlt_init	freezing/melting potential at beginning of time step	W/m^2	
frzmlt_max	maximum magnitude of freezing/melting potential	1000. W/m^2	
frzpnd	• Stefan refreezing of melt ponds	'hlid'	
fsalt	net salt flux to ocean	kg/m^2/s	
fsalt_ai	grid-box-mean salt flux to ocean (fsalt)	kg/m^2/s	
fsens	sensible heat flux	W/m^2	
fsnow	snowfall rate	kg/m^2/s	
fsnowrdg	snow fraction that survives in ridging	0.5	
fsurf(n)(_f)	net surface heat flux excluding fcondtop	W/m^2	
fsw	incoming shortwave radiation	W/m^2	
fswabs	total absorbed shortwave radiation	W/m^2	
fswfac	scaling factor to adjust ice quantities for updated data		
fswint	shortwave absorbed in ice interior	W/m^2	
fswpenl	shortwave penetrating through ice layers	W/m^2	
fswthru	shortwave penetrating to ocean	W/m^2	
fswthru_vdr	visible direct shortwave penetrating to ocean	W/m^2	
fswthru_vdf	visible diffuse shortwave penetrating to ocean	W/m^2	
fswthru_idr	near IR direct shortwave penetrating to ocean	W/m^2	
fswthru_idf	near IR diffuse shortwave penetrating to ocean	W/m^2	
fswthru_ai	grid-box-mean shortwave penetrating to ocean (fswthru)	W/m^2	
fyear	current data year		
fyear_final	last data year		
fyear_init	• initial data year		
G			
gravit	gravitational acceleration	9.80616 m/s^2	
grow_net	specific biogeochemistry growth rate per grid cell	s $^{-1}$	
Gstar	piecewise-linear ridging participation function parameter	0.15	
H			
H2_16O_ocn	concentration of H2_16O isotope in ocean	kg/kg	
H2_18O_ocn	concentration of H2_18O isotope in ocean	kg/kg	
HDO_ocn	concentration of HDO isotope in ocean	kg/kg	
hfrazilmin	minimum thickness of new frazil ice	0.05 m	
hi_min	minimum ice thickness for thinnest ice category	m	
hi_ssl	ice surface scattering layer thickness	0.05 m	
hicen	ice thickness in category n	m	
highfreq	• high-frequency atmo coupling	F	
hin_old	ice thickness prior to growth/melt	m	
hin_max	category thickness limits	m	
history_format	turns on netcdf history output if set to 'nc'		
hmix	ocean mixed layer depth	20. m	
hour	hour of the year		

continues on next page

Table 1 – continued from previous page

hp0	pond depth at which shortwave transition to bare ice occurs	0.2 m	
hp1	• critical ice lid thickness for topo ponds (dEdd)	0.01 m	
hpmin	minimum melt pond depth (shortwave)	0.005 m	
hpondn	melt pond depth	m	
hs_min	minimum thickness for which T_s is computed	$1.\times 10^{-4}$ m	
hs0	• snow depth at which transition to ice occurs (dEdd)		
hs1	• snow depth of transition to pond ice	0.03 m	
hs_ssl	snow surface scattering layer thickness	0.04 m	
Hstar	determines mean thickness of ridged ice	25. m	
I			
i0vis	fraction of penetrating visible solar radiation	0.70	
lateral_flux_type	• laterally flux ice or open water into grid cell when closing		
ice_ic	• choice of initial conditions		
ice_stdout	unit number for standard output		
ice_stderr	unit number for standard error output		
ice_ref_salinity	reference salinity for ice–ocean exchanges	4. ppt	
iceruf	ice surface roughness	$5.\times 10^{-4}$ m	
iceruf_ocn	under-ice roughness	0.03 m	
idate	the date at the end of the current time step (yyyymmdd)		
idate0	initial date		
igrid	interface points for vertical bio grid		
int_kind	definition of an 4-byte integer	selected_int_kind(6)	
int8_kind	definition of an 8-byte integer	selected_int_kind(13)	
istep	local step counter for time loop		
istep0	• number of steps taken in previous run	0	
istep1	total number of steps at current time step		
Iswabs	shortwave radiation absorbed in ice layers	W/m^2	
J			
K			
kalg	• absorption coefficient for algae		
kappav	visible extinction coefficient in ice, wavelength<700nm	1.4 m^{-1}	
kcatbound	• category boundary formula		
kg_to_g	kg to g conversion factor	1000.	
kice	thermal conductivity of fresh ice ([6])	2.03 W/m/deg	
kitd	• type of itd conversions (0 = delta function, 1 = linear remap)	1	
krdg_partic	• ridging participation function	1	
krdg_redist	• ridging redistribution function	1	
krdgn	mean ridge thickness per thickness of ridging ice		
ksno	thermal conductivity of snow	0.30 W/m/deg	
kstrength	• ice stength formulation (1= [57], 0 = [21])	1	

continues on next page

Table 1 – continued from previous page

ktherm	• thermodynamic formulation (-1 none, 1 = [6], 2 = mushy)		
L			
l_brine	flag for brine pocket effects		
l_mpond_fresh	• if true, retain (topo) pond water until ponds drain		
Lfresh	latent heat of melting of fresh ice = Lsub - Lvap	J/kg	
lhcoef	transfer coefficient for latent heat		
lmask_n(s)	northern (southern) hemisphere mask		
log_kind	definition of a logical variable	kind(.true.)	
Lsub	latent heat of sublimation for fresh water	2.835×10^6 J/kg	
Lvap	latent heat of vaporization for fresh water	2.501×10^6 J/kg	
M			
m_to_cm	meters to cm conversion	100.	
m1	constant for lateral melt rate	1.6×10^{-6} m/s deg^{-m2}	
m2	constant for lateral melt rate	1.36	
m2_to_km2	m^2 to km^2 conversion	1×10^{-6}	
max_blocks	maximum number of blocks per processor		
max_ntrcr	maximum number of tracers available	5	
maxraft	maximum thickness of ice that rafts	1. m	
mday	day of the month		
meltb	basal ice melt	m	
meltl	lateral ice melt	m	
melts	snow melt	m	
meltt	top ice melt	m	
min_salin	threshold for brine pockets	0.1 ppt	
mlt_onset	day of year that surface melt begins		
month	the month number		
monthp	previous month number		
mps_to_cmpdy	m per s to cm per day conversion	8.64×10^6	
mu_rdg	• e-folding scale of ridged ice		
N			
n_aero	number of aerosol species		
n_iso	number of water isotope species		
natmiter	• number of atmo boundary layer iterations	5	
nbtrcr	number of biology tracers		
ncat	number of ice categories	5	
ndtd	• number of dynamics/advection steps under thermo	1	
new_day	flag for beginning new day		
new_hour	flag for beginning new hour		
new_month	flag for beginning new month		
new_year	flag for beginning new year		
nfreq	number of wave frequency bins	25	
nfsd	number of floe size categories	12	
nhlat	northern latitude of artificial mask edge	30°S	
nilyr	number of ice layers in each category	7	

continues on next page

Table 1 – continued from previous page

nit	nitrate concentration	mmol/m^3	
nlt_bgc_[chem]	ocean sources and sinks for biogeochemistry		
nml_filename	namelist file name		
npt	• total number of time steps (dt)		
nslyr	number of snow layers in each category		
nspint	number of solar spectral intervals		
nt_<trcr>	tracer index		
ntrcr	number of tracers		
nu_diag	unit number for diagnostics output file		
nu_dump	unit number for dump file for restarting		
nu_forcing	unit number for forcing data file		
nu_nml	unit number for namelist input file		
nu_restart	unit number for restart input file		
nu_rst_pointer	unit number for pointer to latest restart file		
nx(y)_block	total number of gridpoints on block in x(y) direction		
nyr	year number		
O			
ocean_bio	concentrations of bgc constituents in the ocean		
oceanmixed_ice	• if true, use internal ocean mixed layer		
ocn_data_format	• format of ocean forcing files		
ocn_data_type	• source of ocean surface data		
omega	angular velocity of Earth	7.292×10^{-5} rad/s	
opening	rate of ice opening due to divergence and shear	1/s	
P			
p001	1/1000		
p01	1/100		
p025	1/40		
p027	1/36		
p05	1/20		
p055	1/18		
p1	1/10		
p111	1/9		
p15	15/100		
p166	1/6		
p2	1/5		
p222	2/9		
p25	1/4		
p333	1/3		
p4	2/5		
p5	1/2		
p52083	25/48		
p5625m	-9/16		
p6	3/5		
p666	2/3		
p75	3/4		
phi_c_slow_mode	• critical liquid fraction		
phi_i_mushy	• solid fraction at lower boundary		
phi_sk	skeletal layer porosity		
phi_snow	• snow porosity for brine height tracer		

continues on next page

Table 1 – continued from previous page

pi	π		
pi2	2π		
pih	$\pi/2$		
piq	$\pi/4$		
pndaspect	• aspect ratio of pond changes (depth:area)	0.8	
potT	atmospheric potential temperature	K	
PP_net	total primary productivity per grid cell	mg C/m^2/s	
precip_units	• liquid precipitation data units		
print_points	• if true, print point data	F	
Pstar	ice strength parameter	2.75×10^4N/m^2	
puny	a small positive number	1×10^{-11}	
Q			
Qa	specific humidity at 10 m	kg/kg	
Qa_iso	water isotope specific humidity at 10 m	kg/kg	
qdp	deep ocean heat flux	W/m^2	
qqqice	for saturated specific humidity over ice	1.16378×10^7kg/m^3	
qqqocn	for saturated specific humidity over ocean	6.275724×10^6kg/m^3	
Qref	2m atmospheric reference specific humidity	kg/kg	
Qref_iso	2m atmospheric water isotope reference specific humidity	kg/kg	
R			
R_C2N	algal carbon to nitrate factor	7. mole/mole	
R_gC2molC	mg/mmol carbon	12.01 mg/mole	
R_chl2N	algal chlorophyll to nitrate factor	3. mg/mmol	
R_ice	• parameter for Delta-Eddington ice albedo		
R_pnd	• parameter for Delta-Eddington pond albedo		
R_S2N	algal silicate to nitrate factor	0.03 mole/mole	
R_snw	• parameter for Delta-Eddington snow albedo		
r16_kind	definition of quad precision	selected_real_kind(33,4931)	
Rac_rapid_mode	• critical Rayleigh number	10	
rdg_conv	convergence for ridging	1/s	
rdg_shear	shear for ridging	1/s	
real_kind	definition of single precision real	selected_real_kind(6)	
refindx	refractive index of sea ice	1.310	
restart	• if true, initialize using restart file instead of defaults	T	
restart_age	• if true, read age restart file		
restart_bgc	• if true, read bgc restart file		
restart_dir	• path to restart/dump files		
restart_file	• restart file prefix		
restart_format	history files are read/written in binary or netcdf format if set to 'bin' or 'nc' respectively	bin	
restart_[tracer]	• if true, read tracer restart file		
restore_bgc	• if true, restore nitrate/silicate to data		
restore_ice	• if true, restore ice state along lateral boundaries		

continues on next page

Table 1 – continued from previous page

restore_ocn	• restore sst to data		
rfracmin	• minimum melt water fraction added to ponds	0.15	
rfracmax	• maximum melt water fraction added to ponds	1.0	
rhoa	air density	kg/m^3	
rhofresh	density of fresh water	1000.0 kg/m^3	
rhoi	density of ice	917. kg/m^3	
rhos	density of snow	330. kg/m^3	
rhosi	average sea ice density (for hbrine tracer)	940. kg/m^3	
rhow	density of seawater	1026. kg/m^3	
rnilyr	real(nlyr)		
rside	fraction of ice that melts laterally		
rsnw_fresh	freshly fallen snow grain radius	100. $\times 10^{-6}$ m	
rsnw_mlt	• melting snow grain radius	1000. $\times 10^{-6}$ m	
rsnw_nonmelt	nonmelting snow grain radius	500. $\times 10^{-6}$ m	
rsnw_sig	standard deviation of snow grain radius	250. $\times 10^{-6}$ m	
S			
salinz	ice salinity profile	ppt	
saltflux_option	constant or prognostic salinity fluxes	constant	
saltmax	max salinity, at ice base ([6])	3.2 ppt	
scale_factor	scaling factor for shortwave radiation components		
sec	seconds elasped into idate		
secday	number of seconds in a day	86400.	
shcoef	transfer coefficient for sensible heat		
shear	strain rate II component	1/s	
shlat	southern latitude of artificial mask edge	30°N	
shortwave	• flag for shortwave parameterization (‘ccsm3’ or ‘dEdd’)		
sil	silicate concentration	mmol/m^3	
sk_l	skeletal layer thickness	0.03 m	
snowage_drdt0	snowage table 3D data for drdt0 (10^-6 m/hr)		
snowage_kappa	snowage table 3D data for kappa (10^-6 m)		
snowage_rhos	snowage table dimension data for rhos (kg/m^3)		

continues on next page

Table 1 – continued from previous page

snowage_T	snowage table dimension data for temperature (deg K)		
snowage_tau	snowage table 3D data for tau (10^-6 m)		
snowage_Tgrd	snowage table dimension data for temp gradient (deg K/m)		
snoice	snow-ice formation	m	
snowpatch	length scale for parameterizing nonuniform snow coverage	0.02 m	
skl_bgc	• biogeochemistry on/off		
spval	special value (single precision)	10^{30}	
spval_dbl	special value (double precision)	10^{30}	
ss_tltx(y)	sea surface in the x(y) direction	m/m	
sss	sea surface salinity	ppt	
sst	sea surface temperature	C	
Sswabs	shortwave radiation absorbed in snow layers	W/m^2	
stefan-boltzmann	Stefan-Boltzmann constant	5.67×10^{-8} W/m^{2}K^4	
stop_now	if 1, end program execution		
strairx(y)	stress on ice by air in the x(y)-direction (centered in U cell)	N/m^2	
strairx(y)T	stress on ice by air, x(y)-direction (centered in T cell)	N/m^2	
strax(y)	wind stress components from data	N/m^2	
strength	ice strength	N/m	
stress12	internal ice stress, σ_{12}	N/m	
stressm	internal ice stress, $\sigma_{11} - \sigma_{22}$	N/m	
stressp	internal ice stress, $\sigma_{11} + \sigma_{22}$	N/m	
strintx(y)	divergence of internal ice stress, x(y)	N/m^2	
strocnx(y)	ice–ocean stress in the x(y)-direction (U-cell)	N/m^2	
strocnx(y)T	ice–ocean stress, x(y)-dir. (T-cell)	N/m^2	
strtltx(y)	surface stress due to sea surface slope	N/m^2	
swidf	incoming shortwave radiation, near IR, diffuse	W/m^2	
swidr	incoming shortwave radiation, near IR, direct	W/m^2	
swvdf	incoming shortwave radiation, visible, diffuse	W/m^2	
swvdr	incoming shortwave radiation, visible, direct	W/m^2	
sw_redist	option to redistribute shortwave	.false.	
sw_frac	fraction of redistributed shortwave	0.9	
sw_dtemp	temperature threshold from melting for redistributed shortwave	0.02	
T			
Tair	air temperature at 10 m	K	
tday	absolute day number		
Tf	freezing temperature	C	
Tffresh	freezing temp of fresh ice	273.15 K	
tfrz_option	• form of ocean freezing temperature		
thickness_ocn_layer1	thickness of first ocean level	2.0 m	
thinS	minimum ice thickness for brine tracer		
time	total elapsed time	s	
time_forc	time of last forcing update	s	
Timelt	melting temperature of ice top surface	0. C	

continues on next page

Table 1 – continued from previous page

tscale_pnd_drain	mushy pond macroscopic drainage timescale	10 days	
TLAT	latitude of cell center	radians	
Tliquidus_max	maximum liquidus temperature of mush	0. C	
TLON	longitude of cell center	radians	
tmask	land/boundary mask, thickness (T-cell)		
tmass	total mass of ice and snow	kg/m^2	
Tmin	minimum allowed internal temperature	-100. C	
Tmltz	melting temperature profile of ice		
Tocnfrz	temperature of constant freezing point parameterization	C	
tr_aero	• if true, use aerosol tracers		
tr_iso	• if true, use water isotope tracers		
tr_bgc_[tracer]	• if true, use biogeochemistry tracers		
tr_brine	• if true, use brine height tracer		
tr_FY	• if true, use first-year area tracer		
tr_iage	• if true, use ice age tracer		
tr_lvl	• if true, use level ice area and volume tracers		
tr_pond_lvl	• if true, use level-ice melt pond scheme		
tr_pond_topo	• if true, use topo melt pond scheme		
trcr	ice tracers		
trcr_depend	tracer dependency on basic state variables		
Tref	2m atmospheric reference temperature	K	
trestore	• restoring time scale	days	
Tsf_errmax	max allowed T_{sf} error (thermodynamics)	5.$\times 10^{-4}$deg	
Tsfc(n)	temperature of ice/snow top surface (in category n)	C	
Tsmelt	melting temperature of snow top surface	0. C	
TTTice	for saturated specific humidity over ice	5897.8 K	
TTTocn	for saturated specific humidity over ocean	5107.4 K	
U			
uatm	wind velocity in the x direction	m/s	
umin	min wind speed for turbulent fluxes	1. m/s	
uocn	ocean current in the x-direction	m/s	
update_ocn_f	• if true, include frazil ice fluxes in ocean flux fields		
use_leap_years	• if true, include leap days		
ustar_min	• minimum friction velocity under ice		
uvel	x-component of ice velocity	m/s	
V			
vatm	wind velocity in the y direction	m/s	
vice(n)	volume per unit area of ice (in category n)	m	
vicen_init	ice volume at beginning of timestep	m	
viscosity_dyn	dynamic viscosity of brine	1.79×10^{-3} kg/m/s	
vocn	ocean current in the y-direction	m/s	
vonkar	von Karman constant	0.4	
vraftn	volume of rafted ice	m	

continues on next page

Table 1 – continued from previous page

vrdgn	volume of ridged ice	m	
vsno(n)	volume per unit area of snow (in category n)	m	
vvel	y-component of ice velocity	m/s	
W			
warmice	value for constant albedo parameterization	0.68	
warmsno	value for constant albedo parameterization	0.77	
wave_sig_ht	significant height of waves	m	
wave_spectrum	wave spectrum	m/ ^{2}/s	
wavefreq	wave frequencies	1/s	
wind	wind speed	m/s	
write_restart	if 1, write restart now		
X			
Y			
ycycle	• number of years in forcing data cycle		
yday	day of the year		
year_init	• the initial year		
Z			
zlvl	atmospheric level height for momentum (and scalars if zlvs not present)	m	
zlvs	atmospheric level height for scalars	m	
zref	reference height for stability	10. m	
zTrf	reference height for T_{ref}, Q_{ref}, U_{ref}	2. m	
zvir	gas constant (water vapor)/gas constant (air) - 1	0.606	
Deprecated options and parameters			
heat_capacity	if true, use salinity-dependent thermodynamics	T	
kseaice	thermal conductivity of ice for zero-layer thermodynamics	2.0 W/m/deg	
ktherm	thermodynamic formulation (0 = zero-layer, 1 = [6], 2 = mushy)		
tr_pond_cesm	if true, use CESM melt pond scheme		

6.2 Icepack Tutorial

6.2.1 Learning Goals

In this activity you will clone the Icepack model code from the Consortium GitHub repository to run standalone Icepack simulations. You will also make namelist changes and code modifications for experiments and make some basic plots. If you run into issues, contact dbailey@ucar.edu.

Notes:

- Command line text is shown in highlighted boxes.
- When there is the <X> syntax, you need to fill in your personal information (e.g. a URL or username) for that command but without the angle brackets. Your GitHub and local computer usernames may not be the same, so

check which you need to use.

- There is a lot of documentation.
  - Icepack User Guide, https://cice-consortium-icepack.readthedocs.io/en/latest/index.html
  - CICE-Consortium GitHub Usage Guide, https://github.com/CICE-Consortium/About-Us/wiki/Git-Workflow-Guide
  - CICE and Icepack Resources, https://github.com/CICE-Consortium/About-Us/wiki/Resource-Index

6.2.2 Github One-time Configuration

You need to have your own GitHub account before you can start the following activities, and you should have already forked the Icepack repository. For information about how to set up a GitHub account for the Icepack repository, see the Consortium documentation here, https://github.com/CICE-Consortium/About-Us/wiki/Git-Workflow-Guide. The Consortium recommends that you keep your fork's main branch in sync with the Consortium version and that you always work on branches. This is all documented in the Git-Workflow-Guide linked above.

Note:

- The workflow guide is oriented toward setting up CICE rather than Icepack, but the same workflow applies to Icepack standalone. Icepack can be set up and run as an independent model following the same workflow.

6.2.3 Clone Icepack

Clone your Icepack repository fork (use the URL from your fork) to a local sandbox:

```
mkdir ~/icepack-dirs
cd ~/icepack-dirs
git clone https://github.com/<github-user>/Icepack
```

If you have completed this correctly there should be an "Icepack" directory in the icepack-dirs directory. This is the "sandbox" we will be working in locally on your machine.

Move to the Icepack directory and check which branch you are using. This should be main:

```
cd Icepack
git status
```

Take a minute to orient yourself to the big picture structure of the directories and files in Icepack. The documentation has information about the Icepack *Directory structure*.

Make sure your main is up to date and create a branch. You can also update your fork directly in github by clicking the Sync fork button. If your code is already up to date, you can skip this step:

```
git remote --v  (Check the origin and NO upstream)
git remote add upstream https://github.com/CICE-Consortium/Icepack
git remote --v  (Check upstream has been added)
git pull upstream main
git push origin main
git branch <branchname>
git checkout <branchname>
```

6.2.4 Conda and Laptop One-time Configuration

To build and run Icepack on your laptop, you need to install software via conda. Instructions on how to do that can be found in the Icepack user guide, *Porting to Laptop or Personal Computers*. If you have a Windows machine, we recommend using the Ubuntu Linux application, https://ubuntu.com/desktop/wsl. Make sure to follow the instructions for installing miniconda. If your laptop has a conda environment already installed, you will still need to activate the icepack environment, and you may need to do so using the recommended miniconda distribution. Return here after completing section *Porting to Laptop or Personal Computers* in the documentation. After installing miniconda, the main steps are:

```
cd ~/icepack-dirs/Icepack
conda env create -f configuration/scripts/machines/environment.yml
conda activate icepack
```

Before you can run Icepack, you have to set up a directory structure and download the input and forcing datasets:

```
mkdir -p ~/icepack-dirs/runs ~/icepack-dirs/input ~/icepack-dirs/baseline
cd ~/icepack-dirs/input
curl -O https://zenodo.org/records/3728287/files/Icepack_data-20200326.tar.gz
tar -xzf Icepack_data-20200326.tar.gz
```

You can also run Icepack on an external machine that is supported by the Consortium or to which you have ported the code. In this case, you do not need to port to your laptop.

6.2.5 Set Up an Icepack Simulation

Use the online Icepack documentation and in particular the *Quick Start* and *Running Icepack* sections as guidance and for details on the command line settings:

```
cd ~/icepack-dirs
mkdir cases
cd ~/Icepack
./icepack.setup --case ~/icepack-dirs/cases/icepack_test0 --mach <machine> --env <myenv>
```

Notes:

- If you are doing this in the conda environment, the machine is "conda".
- Similarly, the <myenv> variable is set to the compiler on your machine. For the conda environment, this is "macos" or "linux".

The setup script creates a case consistent with the machine and other defined settings under ~/icepack-dirs/cases/ with the name you selected (icepack_test0). The case directory will contain build and run scripts, a namelist file, and other necessary files. Once the case is set up any of these files can be manually edited to refine the desired configuration.

Move to the new case directory and examine the settings:

```
cd ~/icepack-dirs/cases/icepack_test0
```

Open the **icepack.settings** file and look at it briefly. Note the ICE_CASEDIR (it should match this directory) and the ICE_RUNDIR (where the model will be run and output created). Now look at the default namelist settings in **icepack_in**.

Build the code:

```
./icepack.build
```

The build script basically runs gmake under the covers, but there are a number of other tasks that are handled by the script to make the build more robust. If the build is successful you will see the message "COMPILE SUCCESSFUL" at the bottom of the screen. You can also check the README.case file to check the status.

Submit the job. The submit script just submits the run scripts. Look at both **icepack.run** and **icepack.submit** files to see more details. The out-of-the-box run has default settings for the physics and other options. You can have a look at **icepack_in** and **icepack.settings** to review those settings. Then:

```
./icepack.submit
```

If the run is successful, you will see the message "ICEPACK COMPLETED SUCCESSFULLY" in the icepack run log file. Note that this job runs quickly - you are running a column model with four grid cells!

Look at the output! Go to the ICE_RUNDIR where output was created. A successful model integration will create ice_diag.* files and a file in the "restart" directory called "iced.2016-01-01-00000". The Icepack documentation has more information about *Model output*.

Follow the documentation to create some plots of the output using the tools provided with Icepack (*Test Plotting*). The conda icepack environment must be activated, if it isn't already:

```
cd ~/icepack-dirs/Icepack/configuration/scripts/tests/
conda activate icepack
./timeseries.csh ~/icepack-dirs/runs/icepack_test0/ice_diag.full_ITD
```

Note that you can run the plotting script on any of the four ice_diag.* files. The .png files are created in the ICE_RUNDIR directory. Open the files:

```
cd ~/icepack-dirs/runs/icepack_test0/
open <figurename>.png
```

Or use your file browser to navigate to the directory and double click on the images.

Questions to think about while looking at the output.

- What time period does an out-of-the-box run cover?
- What are the differences between the full_ITD plots and the icefree plots (or any other combination of the ice_diag.* output files)? Which fields are the same? Which are different? Why would this be?
- What happens to ice area and ice thickness around October 1, 2015? Why do you see this signal?
- How does your output compare to the sample output provided for this release? (hint: see the wiki!)

Take a step back and think about all the directories and files you have created. The Icepack "sandbox" was cloned from GitHub and has the actual Icepack code.

- There is a particular case directory for building and launching the code, and some output (e.g. job log) are copied.
- There is a particular run directory for each case. This is where the model is run and big files are found.

6.2.6 Set Up a Longer Run

Once you have had success with the previous step, you can run another, longer experiment to practice some basic changes for Icepack. Go back to your Icepack directory:

```
cd ~/icepack-dirs/Icepack/
```

You need to set up a new out-of-the-box case (icepack_test1):

```
./icepack.setup --case ~/icepack-dirs/cases/icepack_test1 --mach <machine> --env <myenv>
```

Go into the cases/icepack_test1 directory, and build the case. Change the following namelist settings in **icepack_in**,

> npt = 8760

How long is this setting the model to run? Change this to run for 10 years (hint: The timestep is one hour, and there are 24 steps per day, and 365 days per year).

Details about namelist options are in the documentation (*Case Settings, Model Namelist, and CPPs*).

Submit the job. Check the output and think about the following:

- Over what dates did the model run this time?
- What date would the model restart from?

6.2.7 Modify a physics option

Set up another case:

```
./icepack.setup --case ~/icepack-dirs/cases/icepack_test2 --mach <machine> --env <myenv>
```

Build the code.

Change the thermodynamics option from ktherm = 2 to ktherm = 1 in **icepack_in**, and set sw_redist = .true. The intent here is to change the namelist option for the current experiment in the case directory. Think about what would happen if you changed **icepack_in** in the source code before creating the case instead (hint: this experiment should work the same, but what about future experiments?).

Submit the job. Have a look at the output.

- What is different compared to your first run?
- What happens if sw_redist = .false. with ktherm = 1? Why?

6.2.8 Change a Parameter in the Fortran Code

Set up another case:

```
./icepack.setup --case ~/icepack-dirs/cases/icepack_test3 --mach <machine> --env <myenv>
```

Change to the source code directory:

```
cd columnphysics
```

Edit icepack_mechred.F90 to change the line

> fsnowrdg = p5 , & ! snow fraction that survives in ridging

to

> fsnowrdg = c1 , & ! snow fraction that survives in ridging

Build the code and submit the job.

- What is different about this run?
- What do you think the fsnowrdg parameter is doing here?

Revert your code changes:

```
cd ~/Icepack
git status
git checkout columnphysics/icepack_mechred.F90
git status
```

CHAPTER SEVEN

REFERENCES

References

- search

BIBLIOGRAPHY

[1] T.L. Amundrud, H. Malling, and R.G. Ingram. Geometrical constraints on the evolution of ridged sea ice. *J. Geophys. Res. Oceans*, 2004. URL: http://dx.doi.org/10.1029/2003JC002251.

[2] K.C. Armour, C.M. Bitz, L. Thompson, and E.C. Hunke. Controls on Arctic sea ice from first-year and multi-year ice survivability. *J. Climate*, 24:2378–2390, 2011. URL: http://dx.doi.org/10.1175/2010JCLI3823.1.

[3] S.P.S. Arya. A drag partition theory for determining the large-scale roughness parameter and wind stress on the Arctic pack ice. *J. Geophys. Res.*, 80:3447–3454, 1975. URL: http://dx.doi.org/10.1029/JC080i024p03447.

[4] A. Assur. Composition of sea ice and its tensile strength. In *Arctic sea ice; conference held at Easton, Maryland, February 24-27, 1958*, volume 598, pages 106–138. Publs. Natl. Res. Coun. Wash., Washington, D.C., 1958.

[5] C.M. Bitz, M.M. Holland, M. Eby, and A.J. Weaver. Simulating the ice-thickness distribution in a coupled climate model. *J. Geophys. Res. Oceans*, 106:2441–2463, 2001. URL: http://dx.doi.org/10.1029/1999JC000113.

[6] C.M. Bitz and W.H. Lipscomb. An energy-conserving thermodynamic sea ice model for climate study. *J. Geophys. Res. Oceans*, 104(C7):15669–15677, 1999. URL: http://dx.doi.org/10.1029/1999JC900100.

[7] E. Brady, S. Stevenson, D. Bailey, Z. Liu, D. Noone, J. Nusbaumer, B. L. Otto-Bliesner, C. Tabor, R. Tomas, T. Wong, J. Zhang, and J. Zhu. The connected isotopic water cycle in the community earth system model version 1. *J. Adv. Modeling Earth Sys.*, 11(8):2547–2566, 2019. URL: https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2019MS001663.

[8] B.P. Briegleb and B. Light. *A Delta-Eddington multiple scattering parameterization for solar radiation in the sea ice component of the Community Climate System Model*. NCAR Technical Note NCAR/TN-472+STR, National Center for Atmospheric Research, 2007. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/BL_NCAR2007.pdf.

[9] W. D. Collins and coauthors. The Community Climate System Model Version 3 (CCSM3). *J. Climate*, 19:2122–2143, 2006. URL: https://doi.org/10.1175/JCLI3761.1.

[10] C. Dang, C. S. Zender, and M. G. Flanner. Intercomparison and improvement of two-stream shortwave radiative transfer schemes in earth system models for a unified treatment of cryospheric surfaces. *The Cryosphere*, 13:2325–2343, 2019. doi:10.5194/tc-13-2325-2019.

[11] P. Duarte and Coauthors. Sea ice thermohaline dynamics and biogeochemistry in the Arctic Ocean: Empirical and model results. *J. Geophys. Res. Biogeo.*, 122:1632–1654, 2017. URL: http://dx.doi.org/10.1002/2016JG003660.

[12] E.E. Ebert, J.L. Schramm, and J.A. Curry. Disposition of solar radiation in sea ice and the upper ocean. *J. Geophys. Res. Oceans*, 100:15965–15975, 1995. URL: http://dx.doi.org/10.1029/95JC01672.

[13] H. Eicken, T.C. Grenfell, D.K. Perovich, J.A Richter-Menge, and K. Frey. Hydraulic controls of summer Arctic pack ice albedo. *J. Geophys. Res. Oceans*, 2004. URL: http://dx.doi.org/10.1029/2003JC001989.

[14] D.L. Feltham, N. Untersteiner, J.S. Wettlaufer, and M.G. Worster. Sea ice is a mushy layer. *Geophys. Res. Lett.*, 2006. URL: http://dx.doi.org/10.1029/2006GL026290.

[15] G.M. Flato and W.D. Hibler. Ridging and strength in modeling the thickness distribution of Arctic sea ice. *J. Geophys. Res. Oceans*, 100:18611–18626, 1995. URL: http://dx.doi.org/10.1029/95JC02091.

[16] D. Flocco and D.L. Feltham. A continuum model of melt pond evolution on Arctic sea ice. *J. Geophys. Res. Oceans*, 2007. URL: http://dx.doi.org/10.1029/2006JC003836.

[17] D. Flocco, D.L. Feltham, and A.K. Turner. Incorporation of a physically based melt pond scheme into the sea ice component of a climate model. *J. Geophys. Res. Oceans*, 2010. URL: http://dx.doi.org/10.1029/2009JC005568.

[18] D. Flocco, D. Schroeder, D.L. Feltham, and E.C. Hunke. Impact of melt ponds on Arctic sea ice simulations from 1990 to 2007. *J. Geophys. Res. Oceans*, 2012. URL: http://dx.doi.org/10.1029/2012JC008195.

[19] National Centers for Environmental Information. World Ocean Atlas Version 2. *Natl. Ocn. and Atm. Admin*, 2013. URL: https://www.nodc.noaa.gov/OC5/woa13/.

[20] K.M. Golden, H. Eicken, A.L. Heaton, J. Miner, D.J. Pringle, and J. Zhu. Thermal evolution of permeability and microstructure in sea ice. *Geophys. Res. Lett.*, 2007. URL: http://dx.doi.org/10.1029/2007GL030447.

[21] W.D. Hibler. A dynamic thermodynamic sea ice model. *J. Phys. Oceanogr.*, 9:817–846, 1979. URL: http://dx.doi.org/10.1175/1520-0485(1979)009\T1\textless{}0815:ADTSIM\T1\textgreater{}2.0.CO;2.

[22] W.D. Hibler. Modeling a variable thickness sea ice cover. *Mon. Wea. Rev.*, 108:1943–1973, 1980. URL: http://dx.doi.org/10.1175/1520-0493(1980)108\T1\textless{}1943:MAVTSI\T1\textgreater{}2.0.CO;2.

[23] M.M. Holland, D.A. Bailey, B.P. Briegleb, B. Light, and E. Hunke. Improved sea ice shortwave radiation physics in CCSM4: The impact of melt ponds and aerosols on Arctic sea ice. *J. Climate*, 25:1413–1430, 2012. URL: http://dx.doi.org/10.1175/JCLI-D-11-00078.1.

[24] C. Horvat and E. Tziperman. A prognostic model of the sea-ice floe size and thickness distribution. *The Cryosphere*, 9(6):2119–2134, 2015. URL: http://dx.doi.org/10.5194/tc-9-2119-2015.

[25] C. Horvat and E. Tziperman. The evolution of scaling laws in the sea ice floe size distribution. *Journal of Geophysical Research: Oceans*, 122(9):7630–7650, 2017. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016JC012573.

[26] E.C. Hunke and C.M. Bitz. Age characteristics in a multidecadal Arctic sea ice simulation. *J. Geophys. Res. Oceans*, 2009. URL: http://dx.doi.org/10.1029/2008JC005186.

[27] E.C. Hunke, D.A. Hebert, and O. Lecomte. Level-ice melt ponds in the Los Alamos Sea Ice Model, CICE. *Ocean Modelling*, 71:26–42, 2013. URL: http://dx.doi.org/10.1016/j.ocemod.2012.11.008.

[28] N. Jeffery and E.C. Hunke. Modeling the winter-spring transition of first-year ice in the western Weddell Sea. *J. Geophys. Res. Oceans*, 119:5891–5920, 2014. URL: http://dx.doi.org/10.1002/2013JC009634.

[29] N. Jeffery, E.C. Hunke, and S.M. Elliott. Modeling the transport of passive tracers in sea ice. *J. Geophys. Res. Oceans*, 116:2156–2202, 2011. URL: http://dx.doi.org/10.1029/2010JC006527.

[30] M. Jin, C. Deal, J. Wang, K.H. Shin, N. Tanaka, T.E. Whiteledge, S.H. Lee, and R.R. Gradinger. Controls of the landfast ice-ocean ecosystem offshore Barrow, Alaska. *Ann. Glaciol.*, 44:63–72, 2006. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/JDWSTWLG06.pdf.

[31] R.E. Jordan, E.L. Andreas, and A.P. Makshtas. Heat budget of snow-covered sea ice at North Pole 4. *J. Geophys. Res. Oceans*, 104(C4):7785–7806, 1999. URL: http://dx.doi.org/10.1029/1999JC900011.

[32] B.G. Kauffman and W.G. Large. *The CCSM coupler, version 5.0.1*. 2002. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/KL_NCAR2002.pdf.

[33] D. Lavoie, K. Denman, and C. Michel. Modeling ice algal growth and decline in a seasonally ice- covered region of the Arctic (Resolute Passage, Canadian Archipelago). *J. Geophys. Res. Oceans*, 2005. URL: http://dx.doi.org/10.1029/2005JC002922.

[34] O. Lecomte, T. Fichefet, D. Flocco, D. Schroeder, and M. Vancoppenolle. Interactions between wind-blown snow redistribution and melt ponds in a coupled ocean-sea ice model. *Ocean Modelling*, 87:67–80, 2015. URL: https://doi.org/10.1016/j.ocemod.2014.12.003.

[35] O. Lecomte, T. Fichefet, M. Vancoppenolle, F. Domine, F. Massonet, P. Mathiot, S. Morin, and P. Y. Barriat. On the formulation of snow thermal conductivity in large-scale sea ice models. *J. Adv. Modeling Earth Sys.*, 5:542–557, 2013. URL: https://doi.org/10.1002/jame.20039.

[36] R.W. Lindsay. Temporal variability of the energy balance of thick Arctic pack ice. *J. Climate*, 11:313–333, 1998. URL: https://doi.org/10.1175/1520-0442(1998)011\T1\textless{}0313:TVOTEB\T1\textgreater{}2.0.CO;2.

[37] W.H. Lipscomb. *Modeling the Thickness Distribution of Arctic Sea Ice*. Dept. of Atmospheric Sciences University of Washington, Seattle, 1998. PhD thesis. URL: http://hdl.handle.net/1773/10081.

[38] W.H. Lipscomb. Remapping the thickness distribution in sea ice models. *J. Geophys. Res. Oceans*, 106:13989–14000, 2001. URL: http://dx.doi.org/10.1029/2000JC000518.

[39] W.H. Lipscomb, E.C. Hunke, W. Maslowski, and J. Jakacki. Ridging, strength, and stability in high-resolution sea ice models. *J. Geophys. Res. Oceans*, 2007. URL: http://dx.doi.org/10.1029/2005JC003355.

[40] P. Lu, Z. Li, B. Cheng, and M. Leppäranta. A parametrization fo the ice-ocean drag coefficient. *J. Geophys. Res. Oceans*, 2011. URL: http://dx.doi.org/10.1029/2010JC006878.

[41] C. Lüpkes, V.M. Gryanik, J. Hartmann, and E.L. Andreas. A parametrization, based on sea ice morphology, of the neutral atmospheric drag coefficients for weather prediction and climate models. *J. Geophys. Res. Atmos.*, 2012. URL: http://dx.doi.org/10.1029/2012JD017630.

[42] G.A. Maykut. Large-scale heat exchange and ice production in the central Arctic. *J. Geophys. Res. Oceans*, 87:7971–7984, 1982. URL: http://dx.doi.org/10.1029/JC087iC10p07971.

[43] G.A. Maykut and M.G. McPhee. Solar heating of the Arctic mixed layer. *J. Geophys. Res. Oceans*, 100:24691–24703, 1995. URL: http://dx.doi.org/10.1029/95JC02554.

[44] G.A. Maykut and D.K. Perovich. The role of shortwave radiation in the summer decay of a sea ice cover. *J. Geophys. Res. Oceans*, 92:7032–7044, 1987. URL: http://dx.doi.org/10.1029/JC092iC07p07032.

[45] G.A. Maykut and N. Untersteiner. Some results from a time dependent thermodynamic model of sea ice. *J. Geophys. Res.*, 76:1550–1575, 1971. URL: http://dx.doi.org/10.1029/JC076i006p01550.

[46] D. Notz. *Thermodynamic and Fluid-Dynamical Processes in Sea Ice*. University of Cambridge, UK, 2005. PhD thesis. URL: http://ulmss-newton.lib.cam.ac.uk/vwebv/holdingsInfo?bibId=27224.

[47] W. K. Oleson, D. M. Lawrence, G. B. Bonan, M. G. Flanner, E. Kluzek, P. J. Lawrence, S. Levis, S. C. Swenson, and P. E. Thornton. *Technical description of version 4.0 of the Community Land Model (CLM)*. NCAR Technical Note NCAR/TN-478+STR, National Center for Atmospheric Research, 2019. URL: https://opensky.ucar.edu/islandora/object/technotes:493.

[48] N. Ono. Specific heat and heat of fusion of sea ice. In H. Oura, editor, *Physics of Snow and Ice*, volume I, pages 599–610. Institute of Low Temperature Science, Hokkaido, Japan, 1967.

[49] D. K. Perovich and K. F. Jones. The seasonal evolution of sea ice floe size distribution. *J. Geophys. Res. Oceans*, 119(12):8767–8777, 2014. URL: http://doi.wiley.com/10.1002/2014JC010136.

[50] D.J. Pringle, H. Eicken, H.J. Trodahl, and L.G.E. Backstrom. Thermal conductivity of landfast Antarctic and Arctic sea ice. *J. Geophys. Res. Oceans*, 2007. URL: http://dx.doi.org/10.1029/2006JC003641.

[51] L. A. Roach, C. Horvat, S. M. Dean, and C. M. Bitz. An emergent sea ice floe size distribution in a global coupled ocean-sea ice model. *J. Geophys. Res. Oceans*, 123(6):4322–4337, 2018. URL: http://dx.doi.org/10.1029/2017JC013692.

[52] L. A. Roach, M. M. Smith, and S. M. Dean. Quantifying growth of pancake sea ice floes using images from drifting buoys. *J. Geophys. Res. Oceans*, 123(4):2851–2866, 2018. URL: http://doi.wiley.com/10.1002/2017JC013693.

[53] L.A. Roach, C. M. Bitz, C. Horvat, and S. M. Dean. Advances in modelling interactions between sea ice and ocean surface waves. *J. Adv. Modeling Earth Sys.*, 11(12):4167–4181, 2019. URL: http://doi.wiley.com/10.1029/2019MS001836.

[54] A. F. Roberts, E. C. Hunke, S. M. Kamal, W. H. Lipscomb, C. Horvat, and W. Maslowski. A Variational Method for Sea Ice Ridging in Earth System Models. *Journal of Advances in Modeling Earth Systems*, 11:771–805, 2019. doi:10.1029/2018MS001395.

[55] A.F. Roberts, A.P. Craig, W. Maslowski, R. Osinski, A.K. DuVivier, M. Hughes, B. Nijssen, J.J. Cassano, and M. Brunke. Simulating transient ice-ocean Ekman transport in the Regional Arctic System Model and Community Earth System Model. *Ann. Glaciol.*, 56(69):211–228, 2015. URL: http://dx.doi.org/10.3189/2015AoG69A760.

[56] D. A. Rothrock and A. S. Thorndike. Measuring the sea ice floe size distribution. *J. Geophys. Res.*, 89(C4):6477, 1984. URL: http://doi.wiley.com/10.1029/JC089iC04p06477.

[57] D.A. Rothrock. The energetics of plastic deformation of pack ice by ridging. *J. Geophys. Res.*, 80:4514–4519, 1975. URL: http://dx.doi.org/10.1029/JC080i033p04514.

[58] F. Roy, M. Chevallier, G.C. Smith, F. Dupont, G. Garric, J.-F. Lemieux, Y. Lu, and F. Davidson. Arctic sea ice and freshwater sensitivity to the treatment of the atmosphere-ice-ocean surface layer. *J. Geophys. Res. Oceans*, 120(6):4392–4417, 2015. URL: https://doi.org/10.1002/2014JC010677, doi:https://doi.org/10.1002/2014JC010677.

[59] S. Saha, S. Moorthi, X. Wu, J. Wang, S. Nadiga, P. Tripp, D Behringer, Y. Hou, H. Chuang, M. Iredell, M. Ek, J. Meng, R. Yang, M.P. Mendez, H. van den Dool, Q. Zhang, W. Wang, M. Chen, and E. Becker. The NCEP Climate Forecast System Version 2. *J. Climate*, 27:2185–2208, 2014. URL: http://dx.doi.org/10.1175/JCLI-D-12-00823.1.

[60] W. Schwarzacher. Pack ice studies in the Arctic Ocean. *J. Geophys. Res.*, 64:2357–2367, 1959. URL: http://dx.doi.org/10.1029/JZ064i012p02357.

[61] A.J. Semtner. A Model for the Thermodynamic Growth of Sea Ice in Numerical Investigations of Climate. *J. Phys. Oceanogr.*, 6:379–389, 1976. URL: http://dx.doi.org/10.1175/1520-0485(1976)006\T1\textless{}0379:AMFTTG\T1\textgreater{}2.0.CO;2.

[62] H. H. Shen, S. F. Ackley, and M. A. Hopkins. A conceptual model for pancake-ice formation in a wave field. *Annals of Glaciology*, 33(2):361–367, 2001. URL: http://dx.doi.org/10.3189/172756401781818239.

[63] G. Siedler and H. Peters. Physical properties (general) of sea water. In *Landolt-Börnstein: Numerical data and functional relationships in science and technology, New Series V/3a*, pages 233–264. Springer, 1986.

[64] M. Steele. Sea ice melting and floe geometry in a simple ice-ocean model. *J. Geophys. Res. Oceans*, 97:17729–17738, 1992. URL: http://dx.doi.org/10.1029/92JC01755.

[65] M. Sturm, J. Holmgren, and D. K. Perovich. Winter snow cover on the sea ice of the Arctic Ocean at the Surface Heat Budget of the Arctic Ocean (SHEBA): Temporal evolution and spatial variability. *J. Geophys. Res. Oceans*, 2002. URL: https://doi.org/10.1029/2000JC000400.

[66] A. Tagliabue, L. Bopp, and O. Aumont. Evaluating the importance of atmospheric and sedimentary iron sources to Southern Ocean biogeochemistry. *Geophys. Res. Lett.*, 2009. URL: http://dx.doi.org/10.1029/2009GL038914.

[67] P.D. Taylor and D.L. Feltham. A model of melt pond evolution on sea ice. *J. Geophys. Res. Oceans*, 2004. URL: http://dx.doi.org/10.1029/2004JC002361.

[68] A.S. Thorndike, D.A. Rothrock, G.A. Maykut, and R. Colony. The thickness distribution of sea ice. *J. Geophys. Res.*, 80:4501–4513, 1975. URL: http://dx.doi.org/10.1029/JC080i033p04501.

[69] H.J. Trodahl, S.O.F. Wilkinson, M.J. McGuinness, and T.G. Haskeel. Thermal conductivity of sea ice: dependence on temperature and depth. *Geophys. Res. Lett.*, 28:1279–1282, 2001. URL: http://dx.doi.org/10.1029/2000GL012088.

[70] M. Tsamados, D.L. Feltham, D. Schroeder, D. Flocco, S.L. Farrell, N.T. Kurtz, S.W. Laxon, and S. Bacon. Impact of variable atmospheric and oceanic form drag on simulations of Arctic sea ice. *J. Phys. Oceanogr.*, 44:1329–1353, 2014. URL: http://dx.doi.org/10.1175/JPO-D-13-0215.1.

[71] A.K. Turner, E.C. Hunke, and C.M. Bitz. Two modes of sea-ice gravity drainage: a parameterization for large-scale modeling. *J. Geophys. Res. Oceans*, 118:2279–2294, 2013. URL: http://dx.doi.org/10.1002/jgrc.20171.

[72] N. Untersteiner. Calculations of temperature regime and heat budget of sea ice in the Central Arctic. *J. Geophys. Res.*, 69:4755–4766, 1964. URL: http://dx.doi.org/10.1029/JZ069i022p04755.