```
#include "fabm_driver.h"
#include "fabm_private.h"

! ===========================================================================
! FABM --- Framework for Aquatic Biogeochemical Models
! ---------------------------------------------------------------------------
! This is the core module of FABM, serving as the "glue layer" between a
! physical host model (e.g., a general circulation model), and one or more
! specific biogeochemical models. A physical host model will call the
! interfaces of this module to access biogeochemistry.
!
! For more information, see the documentation at http://fabm.net/wiki.
!
! To add new biogeochemical models, add source code under src/models and
! reference your institute in src/CMakeLists.txt
! ===========================================================================

module fabm

   use fabm_parameters
   use fabm_types, rki => rk, fabm_standard_variables => standard_variables
   use fabm_expressions
   use fabm_driver
   use fabm_properties
   use fabm_builtin_depth_integral
   use fabm_builtin_reduction
   use fabm_coupling
   use fabm_job
   use fabm_schedule
   use fabm_debug
   use fabm_work

   implicit none

   private

   ! ---------------------------------------------------------------------------
   ! Public members
   ! ---------------------------------------------------------------------------

   public fabm_initialize_library
   public fabm_get_version
   public fabm_create_model
   public fabm_finalize_library
   public type_fabm_model

   ! Variable identifier types by external physical drivers.
   public type_fabm_variable_id
   public type_fabm_interior_variable_id
   public type_fabm_horizontal_variable_id
   public type_fabm_scalar_variable_id
   public type_fabm_variable, type_fabm_interior_state_variable, type_fabm_horizontal_state_variable, &
      type_fabm_interior_diagnostic_variable, type_fabm_horizontal_diagnostic_variable

   ! Object with all supported standard variables as its members.
   ! Imported from fabm_types, and made available so hosts only need to "use fabm"
   public fabm_standard_variables

   integer, parameter :: status_none             = 0
   integer, parameter :: status_initialize_done  = 1
   integer, parameter, public :: status_set_domain_done  = 2
   integer, parameter, public :: status_start_done = 3

   integer, parameter, public :: data_source_none = 0
   integer, parameter, public :: data_source_host = 1
   integer, parameter, public :: data_source_fabm = 2
   integer, parameter, public :: data_source_user = 3
   integer, parameter, public :: data_source_default = data_source_host

   logical, save :: default_driver = .false.

   ! ---------------------------------------------------------------------------
   ! Derived typed for variable identifiers
   ! ---------------------------------------------------------------------------

   type type_fabm_variable_id
      type (type_internal_variable), pointer :: variable => null()
   end type

   type, extends(type_fabm_variable_id) :: type_fabm_interior_variable_id
   end type

   type, extends(type_fabm_variable_id) :: type_fabm_horizontal_variable_id
   end type

   type, extends(type_fabm_variable_id) :: type_fabm_scalar_variable_id
   end type

   ! ---------------------------------------------------------------------------
   ! Derived types for variable metadata
   ! ---------------------------------------------------------------------------

   ! Derived type for metadata of a generic variable (base type)
   type, abstract :: type_fabm_variable
      character(len=attribute_length) :: name          = ''
      character(len=attribute_length) :: long_name     = ''
      character(len=attribute_length) :: local_long_name = ''
      character(len=attribute_length) :: units         = ''
```

```
 99       character(len=attribute_length) :: path          = ''
100       real(rke)                       :: minimum       = -1.e20_rke
101       real(rke)                       :: maximum       =  1.e20_rke
102       real(rke)                       :: missing_value = -2.e20_rke
103       integer                         :: output        = output_instantaneous ! See output_* parameters defined in fa
    bm_types
104       type (type_property_dictionary) :: properties
105       integer                         :: externalid    = 0                    ! Identifier to be used freely by host
106       type (type_internal_variable), pointer :: target => null()
107    end type
108
109    ! Derived type for interior state variable metadata
110    type, extends(type_fabm_variable) :: type_fabm_interior_state_variable
111       class (type_interior_standard_variable), pointer :: standard_variable => null()
112       real(rke)                                        :: initial_value             = 0.0_rke
113       logical                                          :: no_precipitation_dilution = .false.
114       logical                                          :: no_river_dilution         = .false.
115    end type
116
117    ! Derived type for horizontal (bottom/surface) state variable metadata
118    type, extends(type_fabm_variable) :: type_fabm_horizontal_state_variable
119       class (type_horizontal_standard_variable), pointer :: standard_variable => null()
120       real(rke)                                          :: initial_value = 0.0_rke
121    end type
122
123    ! Derived type for interior diagnostic variable metadata
124    type, extends(type_fabm_variable) :: type_fabm_interior_diagnostic_variable
125       class (type_interior_standard_variable), pointer :: standard_variable => null()
126       logical                                          :: save = .false.
127       integer                                          :: source
128    end type
129
130    ! Derived type for horizontal diagnostic variable metadata
131    type, extends(type_fabm_variable) :: type_fabm_horizontal_diagnostic_variable
132       class (type_horizontal_standard_variable), pointer :: standard_variable => null()
133       logical                                            :: save = .false.
134       integer                                            :: source
135    end type
136
137    ! Derived type for conserved quantity metadata
138    type, extends(type_fabm_variable) :: type_fabm_conserved_quantity
139       class (type_base_standard_variable), pointer :: standard_variable => null()
140       integer                                      :: index             = -1
141       integer                                      :: horizontal_index  = -1
142       type (type_internal_variable),       pointer :: target_hz => null()
143    end type
144
145    ! ----------------------------------------------------------------------
146    ! Derived type for a biogeochemical model as seen by the host
147    ! ----------------------------------------------------------------------
148
149    type type_check_state_data
150       integer   :: index
151       real(rki) :: minimum
152       real(rki) :: maximum
153    end type
154
155    type type_fabm_model
156       ! Variable metadata
157       type (type_fabm_interior_state_variable),        allocatable, dimension(:) :: interior_state_variables
158       type (type_fabm_horizontal_state_variable),      allocatable, dimension(:) :: surface_state_variables
159       type (type_fabm_horizontal_state_variable),      allocatable, dimension(:) :: bottom_state_variables
160       type (type_fabm_interior_diagnostic_variable),   allocatable, dimension(:) :: interior_diagnostic_variables
161       type (type_fabm_horizontal_diagnostic_variable), allocatable, dimension(:) :: horizontal_diagnostic_variables
162       type (type_fabm_conserved_quantity),             allocatable, dimension(:) :: conserved_quantities
163
164       ! Names of variables taken as input by one or more biogeochemical models.
165       ! These may be accessed by the host to enumerate potential forcing variables.
166       character(len=attribute_length), allocatable, dimension(:) :: dependencies
167       character(len=attribute_length), allocatable, dimension(:) :: dependencies_hz
168       character(len=attribute_length), allocatable, dimension(:) :: dependencies_scalar
169
170       ! Individual jobs
171       type (type_job) :: get_interior_sources_job
172       type (type_job) :: get_bottom_sources_job
173       type (type_job) :: get_surface_sources_job
174       type (type_job) :: get_vertical_movement_job
175       type (type_job) :: get_interior_conserved_quantities_job
176       type (type_job) :: get_horizontal_conserved_quantities_job
177       type (type_job) :: finalize_outputs_job
178       type (type_job) :: prepare_inputs_job
179       type (type_job) :: check_interior_state_job
180       type (type_job) :: check_bottom_state_job
181       type (type_job) :: check_surface_state_job
182       type (type_job) :: initialize_interior_state_job
183       type (type_job) :: initialize_bottom_state_job
184       type (type_job) :: initialize_surface_state_job
185
186       ! Root container of biogeochemical modules
187       type (type_base_model) :: root
188
189       integer :: status = status_none
190       logical :: log = .false.
191
192       type (type_link_list) :: links_postcoupling
193
194       type (type_global_variable_register) :: variable_register
195       type (type_job_manager)              :: job_manager
```

```
196       type (type_catalog)                 :: catalog
197       type (type_store)                   :: store
198       type (type_schedules)               :: schedules
199       type (type_domain)                  :: domain
200
201       ! Memory caches for exchanging information with individual biogeochemical modules
202       type (type_interior_cache)   :: cache_int
203       type (type_horizontal_cache) :: cache_hz
204       type (type_vertical_cache)   :: cache_vert
205
206       ! Cache fill values
207       type (type_cache_fill_values) :: cache_fill_values
208
209       ! Information for check_state routines
210       type (type_check_state_data), allocatable, private :: check_interior_state_data(:)
211       type (type_check_state_data), allocatable, private :: check_surface_state_data(:)
212       type (type_check_state_data), allocatable, private :: check_bottom_state_data(:)
213    contains
214       procedure :: initialize
215       procedure :: finalize
216       procedure :: set_domain
217 #if _FABM_DIMENSION_COUNT_>0
218       procedure :: set_domain_start
219       procedure :: set_domain_stop
220 #endif
221 #if defined(_FABM_DEPTH_DIMENSION_INDEX_)&&_FABM_BOTTOM_INDEX_==-1
222       procedure :: set_bottom_index
223 #endif
224 #ifdef _HAS_MASK_
225       procedure :: set_mask
226 #endif
227       procedure :: start
228
229       procedure :: initialize_interior_state
230       procedure :: initialize_bottom_state
231       procedure :: initialize_surface_state
232
233       procedure :: check_interior_state
234       procedure :: check_bottom_state
235       procedure :: check_surface_state
236
237       procedure :: prepare_inputs1
238       procedure :: prepare_inputs2
239       generic :: prepare_inputs => prepare_inputs1, prepare_inputs2
240       procedure :: finalize_outputs
241
242       procedure :: get_interior_sources_rhs
243       procedure :: get_interior_sources_ppdd
244       generic :: get_interior_sources => get_interior_sources_rhs, get_interior_sources_ppdd
245       procedure :: get_bottom_sources_rhs
246       procedure :: get_bottom_sources_ppdd
247       generic :: get_bottom_sources => get_bottom_sources_rhs, get_bottom_sources_ppdd
248       procedure :: get_surface_sources
249
250       procedure :: get_vertical_movement
251       procedure :: get_interior_conserved_quantities
252       procedure :: get_horizontal_conserved_quantities
253
254       procedure :: link_interior_data_by_variable
255       procedure :: link_interior_data_by_id
256       procedure :: link_interior_data_by_sn
257       procedure :: link_interior_data_by_name
258       generic :: link_interior_data => link_interior_data_by_variable, link_interior_data_by_id, link_interior_data_b
    y_sn, link_interior_data_by_name
259
260       procedure :: link_horizontal_data_by_variable
261       procedure :: link_horizontal_data_by_id
262       procedure :: link_horizontal_data_by_sn
263       procedure :: link_horizontal_data_by_name
264       generic :: link_horizontal_data => link_horizontal_data_by_variable, link_horizontal_data_by_id, link_horizonta
    l_data_by_sn, link_horizontal_data_by_name
265
266       procedure :: link_scalar_by_variable
267       procedure :: link_scalar_by_id
268       procedure :: link_scalar_by_sn
269       procedure :: link_scalar_by_name
270       generic :: link_scalar => link_scalar_by_variable, link_scalar_by_id, link_scalar_by_sn, link_scalar_by_name
271
272       procedure :: link_interior_state_data
273       procedure :: link_bottom_state_data
274       procedure :: link_surface_state_data
275       procedure :: link_all_interior_state_data
276       procedure :: link_all_bottom_state_data
277       procedure :: link_all_surface_state_data
278
279       procedure :: require_interior_data
280       procedure :: require_horizontal_data
281       generic :: require_data => require_interior_data, require_horizontal_data
282
283       procedure :: get_interior_data
284       procedure :: get_horizontal_data
285       procedure :: get_scalar_data
286       generic :: get_data => get_interior_data, get_horizontal_data, get_scalar_data
287
288       procedure :: get_interior_diagnostic_data
289       procedure :: get_horizontal_diagnostic_data
290
291       procedure :: get_interior_variable_id_by_name
```

```
292       procedure :: get_interior_variable_id_sn
293       generic :: get_interior_variable_id => get_interior_variable_id_by_name, get_interior_variable_id_sn
294
295       procedure :: get_horizontal_variable_id_by_name
296       procedure :: get_horizontal_variable_id_sn
297       generic :: get_horizontal_variable_id => get_horizontal_variable_id_by_name, get_horizontal_variable_id_sn
298
299       procedure :: get_scalar_variable_id_by_name
300       procedure :: get_scalar_variable_id_sn
301       generic :: get_scalar_variable_id => get_scalar_variable_id_by_name, get_scalar_variable_id_sn
302
303       procedure, nopass :: is_variable_used
304       procedure :: get_variable_name
305
306       procedure :: interior_variable_needs_values
307       procedure :: interior_variable_needs_values_sn
308       procedure :: horizontal_variable_needs_values
309       procedure :: horizontal_variable_needs_values_sn
310       procedure :: scalar_variable_needs_values
311       procedure :: scalar_variable_needs_values_sn
312       generic :: variable_needs_values => interior_variable_needs_values, interior_variable_needs_values_sn, &
313                                           horizontal_variable_needs_values, horizontal_variable_needs_values_sn, &
314                                           scalar_variable_needs_values, scalar_variable_needs_values_sn
315
316       procedure :: process_job
317       generic :: process => process_job
318 #if _FABM_DIMENSION_COUNT_ > 1 || (_FABM_DIMENSION_COUNT_ == 1 && !defined(_FABM_DEPTH_DIMENSION_INDEX_))
319       procedure :: process_job_everywhere
320       generic :: process => process_job_everywhere
321 #endif
322
323    end type type_fabm_model
324
325 contains
326
327    ! ----------------------------------------------------------------------
328    ! fabm_initialize_library: initialize FABM library
329    ! ----------------------------------------------------------------------
330    ! This will be called automatically when creating new models.
331    ! For instance, from fabm_create_model.
332    ! ----------------------------------------------------------------------
333    subroutine fabm_initialize_library()
334       use fabm_library, only: fabm_model_factory
335
336       ! Do nothing if already initialized.
337       if (associated(factory)) return
338
339       ! If needed, create default object for communication (e.g., logging, error reporting) with host.
340       if (.not. associated(driver)) then
341          allocate(driver)
342          default_driver = .true.
343       end if
344
345       ! Create all standard variable objects.
346       call fabm_standard_variables%initialize()
347
348       ! Create the model factory.
349       factory => fabm_model_factory
350       call factory%initialize()
351    end subroutine fabm_initialize_library
352
353    ! ----------------------------------------------------------------------
354    ! fabm_finalize_library: finalize FABM library
355    ! ----------------------------------------------------------------------
356    ! This deallocates all global variables created by fabm_initialize_library
357    ! ----------------------------------------------------------------------
358    subroutine fabm_finalize_library()
359       call fabm_standard_variables%finalize()
360
361       if (associated(driver) .and. default_driver) deallocate(driver)
362       if (associated(factory)) call factory%finalize()
363       factory => null()
364    end subroutine fabm_finalize_library
365
366    ! ----------------------------------------------------------------------
367    ! fabm_get_version: get FABM version string
368    ! ----------------------------------------------------------------------
369    subroutine fabm_get_version(string)
370       use fabm_version
371
372       character(len=*), intent(out) :: string
373
374       type (type_version), pointer :: version
375
376       call fabm_initialize_library()
377       string = git_commit_id // ' (' // git_branch_name // ' branch)'
378       version => first_module_version
379       do while (associated(version))
380          string = trim(string) // ', ' // trim(version%module_name) // ': ' // trim(version%version_string)
381          version => version%next
382       end do
383    end subroutine fabm_get_version
384
385    ! ----------------------------------------------------------------------
386    ! fabm_create_model: create a model from a yaml-based configuration file
387    ! ----------------------------------------------------------------------
388    function fabm_create_model(path, initialize, parameters, unit) result(model)
389       use fabm_config, only: fabm_configure_model
```

```
      character(len=*),               optional, intent(in) :: path
      logical,                        optional, intent(in) :: initialize
      type (type_property_dictionary), optional, intent(in) :: parameters
      integer,                        optional, intent(in) :: unit
      class (type_fabm_model), pointer                     :: model

      logical :: initialize_

      ! Make sure the library is initialized.
      call fabm_initialize_library()

      allocate(model)
      call fabm_configure_model(model%root, model%schedules, model%log, path, parameters=parameters, unit=unit)

      ! Initialize model tree
      initialize_ = .true.
      if (present(initialize)) initialize_ = initialize
      if (initialize_) call model%initialize()
   end function fabm_create_model

   ! ----------------------------------------------------------------------
   ! initialize: initialize a model object
   ! ----------------------------------------------------------------------
   ! This freezes the tree of biogeochemical modules; afterwards no new modules
   ! can be added. This routine will be called automatically from
   ! fabm_create_model unless called with initialize=.false.
   ! ----------------------------------------------------------------------
   subroutine initialize(self)
      class (type_fabm_model), target, intent(inout) :: self

      class (type_property), pointer :: property => null()
      integer                        :: islash
      integer                        :: ivar

      if (self%status >= status_initialize_done) &
         call fatal_error('initialize', 'initialize has already been called on this model object.')

      ! Create zero fields.
      call self%root%add_interior_variable('zero', act_as_state_variable=.true., source=source_constant, missing_value=0.0_rki, output=output_none)
      call self%root%add_horizontal_variable('zero_hz', act_as_state_variable=.true., source=source_constant, missing_value=0.0_rki, output=output_none)

      ! Filter out expressions that FABM can handle itself.
      ! The remainder, if any, must be handled by the host model.
      call filter_expressions(self)

      ! This will resolve all FABM dependencies and generate final authoritative lists of variables of different types.
      call freeze_model_info(self%root)

      ! Raise error for unused coupling commands.
      property => self%root%couplings%first
      do while (associated(property))
         if (.not.self%root%couplings%retrieved%contains(trim(property%name))) then
            islash = index(property%name, '/', .true.)
            call fatal_error('initialize', 'model ' // property%name(1:islash-1) // ' does not contain variable "' // trim(property%name(islash+1:)) // '" mentioned in coupling section.')
         end if
         property => property%next
      end do

      ! Build final authoritative arrays with variable metadata.
      call classify_variables(self)

      ! Create catalog for storing pointers to data per variable.
      call create_catalog(self)

      ! Create built-in jobs, which can then be chained by the host/user by calling job%set_next.
      ! (the reason for chaining is to allow later jobs to use results of earlier ones, thus reducing the number of calls needed)
      call self%job_manager%create(self%prepare_inputs_job, 'prepare_inputs')
      call self%job_manager%create(self%get_interior_sources_job, 'get_interior_sources', source=source_do, previous=self%prepare_inputs_job)
      call self%job_manager%create(self%get_surface_sources_job, 'get_surface_sources', source=source_do_surface, previous=self%prepare_inputs_job)
      call self%job_manager%create(self%get_bottom_sources_job, 'get_bottom_sources', source=source_do_bottom, previous=self%prepare_inputs_job)
      call self%job_manager%create(self%get_interior_conserved_quantities_job, 'get_interior_conserved_quantities', source=source_do, previous=self%prepare_inputs_job)
      call self%job_manager%create(self%get_horizontal_conserved_quantities_job, 'get_horizontal_conserved_quantities', source=source_do_horizontal, previous=self%prepare_inputs_job)
      call self%job_manager%create(self%finalize_outputs_job, 'finalize_outputs', outsource_tasks=.true.)
      call self%get_interior_sources_job%connect(self%finalize_outputs_job)
      call self%get_surface_sources_job%connect(self%finalize_outputs_job)
      call self%get_bottom_sources_job%connect(self%finalize_outputs_job)
      !call self%get_interior_conserved_quantities_job%connect(self%finalize_outputs_job)
      !call self%get_horizontal_conserved_quantities_job%connect(self%finalize_outputs_job)
      call self%job_manager%create(self%get_vertical_movement_job, 'get_vertical_movement', source=source_get_vertical_movement, previous=self%finalize_outputs_job)
      call self%job_manager%create(self%initialize_interior_state_job, 'initialize_interior_state', source=source_initialize_state, previous=self%finalize_outputs_job)
      call self%job_manager%create(self%initialize_bottom_state_job, 'initialize_bottom_state', source=source_initialize_bottom_state, previous=self%finalize_outputs_job)
      call self%job_manager%create(self%initialize_surface_state_job, 'initialize_surface_state', source=source_initialize_surface_state, previous=self%finalize_outputs_job)
      call self%job_manager%create(self%check_interior_state_job, 'check_interior_state', source=source_check_state, previous=self%finalize_outputs_job)
```

```
473       call self%job_manager%create(self%check_bottom_state_job, 'check_bottom_state', source=source_check_bottom_stat
    e, previous=self%finalize_outputs_job)
474       call self%job_manager%create(self%check_surface_state_job, 'check_surface_state', source=source_check_surface_s
    tate, previous=self%finalize_outputs_job)
475
476       call require_flux_computation(self%get_bottom_sources_job, self%links_postcoupling, domain_bottom)
477       call require_flux_computation(self%get_surface_sources_job, self%links_postcoupling, domain_surface)
478       call require_flux_computation(self%get_interior_sources_job, self%links_postcoupling, domain_interior)
479       call require_flux_computation(self%get_vertical_movement_job, self%links_postcoupling, domain_interior + 999)
480
481       call require_call_all_with_state(self%initialize_interior_state_job, self%root%links, domain_interior, source_i
    nitialize_state)
482       call require_call_all_with_state(self%initialize_bottom_state_job, self%root%links, domain_bottom, source_initi
    alize_bottom_state)
483       call require_call_all_with_state(self%initialize_surface_state_job, self%root%links, domain_surface, source_ini
    tialize_surface_state)
484       call require_call_all_with_state(self%check_interior_state_job, self%root%links, domain_interior, source_check_
    state)
485       call require_call_all_with_state(self%check_bottom_state_job, self%root%links, domain_bottom, source_check_bott
    om_state)
486       call require_call_all_with_state(self%check_bottom_state_job, self%root%links, domain_interior, source_check_bo
    ttom_state)
487       call require_call_all_with_state(self%check_surface_state_job, self%root%links, domain_surface, source_check_su
    rface_state)
488       call require_call_all_with_state(self%check_surface_state_job, self%root%links, domain_interior, source_check_s
    urface_state)
489
490       do ivar = 1, size(self%interior_state_variables)
491          call self%check_interior_state_job%read_cache_loads%add(self%interior_state_variables(ivar)%target)
492       end do
493       do ivar = 1, size(self%bottom_state_variables)
494          call self%check_bottom_state_job%read_cache_loads%add(self%bottom_state_variables(ivar)%target)
495       end do
496       do ivar = 1, size(self%surface_state_variables)
497          call self%check_surface_state_job%read_cache_loads%add(self%surface_state_variables(ivar)%target)
498       end do
499
500       do ivar = 1, size(self%conserved_quantities)
501          call self%get_interior_conserved_quantities_job%request_variable(self%conserved_quantities(ivar)%target)
502          call self%get_horizontal_conserved_quantities_job%request_variable(self%conserved_quantities(ivar)%target_hz
    )
503          call self%conserved_quantities(ivar)%target%write_indices%append(self%conserved_quantities(ivar)%index)
504          call self%conserved_quantities(ivar)%target_hz%write_indices%append(self%conserved_quantities(ivar)%horizont
    al_index)
505       end do
506
507       self%status = status_initialize_done
508    end subroutine initialize
509
510    ! ----------------------------------------------------------------------
511    ! finalize: deallocate model object
512    ! ----------------------------------------------------------------------
513    subroutine finalize(self)
514       class (type_fabm_model), target, intent(inout) :: self
515       self%status = status_none
516
517       call self%job_manager%finalize()
518       call self%variable_register%finalize()
519       call self%root%finalize()
520       call self%links_postcoupling%finalize()
521    end subroutine finalize
522
523    ! ----------------------------------------------------------------------
524    ! set_domain: set extents of spatial domain and optionally time step length
525    ! ----------------------------------------------------------------------
526    ! The time step length, seconds_per_time_unit, is the scale factor that
527    ! converts the time value provided to prepare_inputs to seconds.
528    ! The combination of this scale factor and the time value allows
529    ! FABM to determine the number of seconds that has passed between calls
530    ! to prepare_inputs. In turn this enables calculation of moving averages
531    ! ----------------------------------------------------------------------
532    subroutine set_domain(self _POSTARG_LOCATION_, seconds_per_time_unit)
533       class (type_fabm_model), target, intent(inout) :: self
534       _DECLARE_ARGUMENTS_LOCATION_
535       real(rke), optional,             intent(in)    :: seconds_per_time_unit
536
537       class (type_expression), pointer :: expression
538
539       if (self%status < status_initialize_done) call fatal_error('set_domain', 'initialize has not yet been called on
     this model object.')
540       if (self%status >= status_set_domain_done) call fatal_error('set_domain', 'set_domain has already been called o
    n this model object.')
541       self%status = status_set_domain_done
542
543 #if _FABM_DIMENSION_COUNT_>0
544       self%domain%shape(:) = (/_LOCATION_/)
545       self%domain%start(:) = 1
546       self%domain%stop(:) = self%domain%shape
547 #endif
548 #if _HORIZONTAL_DIMENSION_COUNT_>0
549       self%domain%horizontal_shape(:) = (/_HORIZONTAL_LOCATION_/)
550 #endif
551
552       if (present(seconds_per_time_unit)) then
553          ! Since the host provides information about time, we will support time filters.
554          ! These includes moving average and moving maximum filters.
555          expression => self%root%first_expression
556          do while (associated(expression))
```

```
557             select type (expression)
558             class is (type_interior_temporal_mean)
559                ! Moving average of interior variable
560                expression%in = expression%link%target%catalog_index
561                expression%period = expression%period / seconds_per_time_unit
562                allocate(expression%history(_PREARG_LOCATION_ expression%n + 1))
563                expression%history = 0.0_rke
564 #if _FABM_DIMENSION_COUNT_>0
565                allocate(expression%previous_value _INDEX_LOCATION_, expression%last_exact_mean _INDEX_LOCATION_, expr
    ession%mean _INDEX_LOCATION_)
566 #endif
567                expression%last_exact_mean = 0.0_rke
568                expression%mean = expression%missing_value
569                call self%link_interior_data(expression%output_name, expression%mean)
570             class is (type_horizontal_temporal_mean)
571                ! Moving average of horizontal variable
572                expression%in = expression%link%target%catalog_index
573                expression%period = expression%period / seconds_per_time_unit
574                allocate(expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 3))
575                expression%history = 0.0_rke
576                call self%link_horizontal_data(expression%output_name, &
577                                               expression%history(_PREARG_HORIZONTAL_LOCATION_DIMENSIONS_ expression%n
     + 3))
578             class is (type_horizontal_temporal_maximum)
579                ! Moving maximum of horizontal variable
580                expression%in = expression%link%target%catalog_index
581                expression%period = expression%period / seconds_per_time_unit
582                allocate(expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n))
583                expression%history = -huge(1.0_rke)
584 #if _HORIZONTAL_DIMENSION_COUNT_>0
585                allocate(expression%previous_value _INDEX_HORIZONTAL_LOCATION_, expression%maximum _INDEX_HORIZONTAL_L
    OCATION_)
586 #endif
587                expression%maximum = expression%missing_value
588                call self%link_horizontal_data(expression%output_name, expression%maximum)
589             end select
590             expression => expression%next
591          end do
592       end if
593    end subroutine set_domain
594
595 #if _FABM_DIMENSION_COUNT_>0
596    ! ----------------------------------------------------------------------
597    ! set_domain_start: set start index of all spatial dimensions
598    ! ----------------------------------------------------------------------
599    ! This is optional; by default the start index for all dimensions is 1.
600    ! ----------------------------------------------------------------------
601    subroutine set_domain_start(self _POSTARG_LOCATION_)
602       class (type_fabm_model), target, intent(inout) :: self
603       _DECLARE_ARGUMENTS_LOCATION_
604       if (self%status < status_set_domain_done) &
605          call fatal_error('set_domain_start', 'set_domain has not yet been called on this model object.')
606       self%domain%start(:) = (/_LOCATION_/)
607    end subroutine set_domain_start
608
609    ! ----------------------------------------------------------------------
610    ! set_domain_stop: set stop index of all spatial dimensions (default=domain size)
611    ! ----------------------------------------------------------------------
612    ! This is optional; by default the stop index for all dimensions matches
613    ! the domain size provided to set_domain.
614    ! ----------------------------------------------------------------------
615    subroutine set_domain_stop(self _POSTARG_LOCATION_)
616       class (type_fabm_model), target, intent(inout) :: self
617       _DECLARE_ARGUMENTS_LOCATION_
618       if (self%status < status_set_domain_done) &
619          call fatal_error('set_domain_stop', 'set_domain has not yet been called on this model object.')
620       self%domain%stop(:) = (/_LOCATION_/)
621    end subroutine set_domain_stop
622 #endif
623
624 #ifdef _HAS_MASK_
625    ! ----------------------------------------------------------------------
626    ! set_mask: provide spatial mask
627    ! ----------------------------------------------------------------------
628    ! As FABM will keep a pointer to the mask, it needs to remain valid for
629    ! the lifetime of the model object.
630    ! ----------------------------------------------------------------------
631 #  ifdef _FABM_HORIZONTAL_MASK_
632    subroutine set_mask(self, mask_hz)
633 #  else
634    subroutine set_mask(self, mask, mask_hz)
635 #  endif
636       class (type_fabm_model), target, intent(inout)                     :: self
637 #  ifndef _FABM_HORIZONTAL_MASK_
638       _FABM_MASK_TYPE_, target, intent(in) _ATTRIBUTES_GLOBAL_           :: mask
639 #  endif
640       _FABM_MASK_TYPE_, target, intent(in) _ATTRIBUTES_GLOBAL_HORIZONTAL_ :: mask_hz
641
642       integer :: i
643
644       if (self%status < status_set_domain_done) &
645          call fatal_error('set_mask', 'set_domain has not yet been called on this model object.')
646
647 #  ifndef _FABM_HORIZONTAL_MASK_
648 #    if !defined(NDEBUG)&&_FABM_DIMENSION_COUNT_>0
649       do i = 1, size(self%domain%shape)
650          if (size(mask, i) /= self%domain%shape(i)) &
651             call fatal_error('set_mask', 'shape of provided mask does not match domain extents provided to set_domain
```

```
.')
      end do
#    endif
      self%domain%mask => mask
#  endif

#  if !defined(NDEBUG)&&_HORIZONTAL_DIMENSION_COUNT_>0
      do i = 1, size(self%domain%horizontal_shape)
         if (size(mask_hz, i) /= self%domain%horizontal_shape(i)) &
            call fatal_error('set_mask', 'shape of provided horizontal mask does not match domain extents provided to set_domain.')
      end do
#  endif
      self%domain%mask_hz => mask_hz

   end subroutine set_mask
#endif

#if defined(_FABM_DEPTH_DIMENSION_INDEX_)&&_FABM_BOTTOM_INDEX_==-1
   ! ----------------------------------------------------------------------
   ! set_bottom_index: provide bottom indices for every horizontal point
   ! ----------------------------------------------------------------------
   ! As FABM will keep a pointer to the array with indices, it needs to remain
   ! valid for the lifetime of the model object.
   ! ----------------------------------------------------------------------
   subroutine set_bottom_index(self, indices)
      class (type_fabm_model), intent(inout)                            :: self
      integer, target,         intent(in) _ATTRIBUTES_GLOBAL_HORIZONTAL_ :: indices

      integer :: i

      if (self%status < status_set_domain_done) &
         call fatal_error('set_bottom_index', 'set_domain has not yet been called on this model object.')
#  if !defined(NDEBUG)&&_HORIZONTAL_DIMENSION_COUNT_>0
      do i = 1, size(self%domain%horizontal_shape)
         if (size(indices, i) /= self%domain%horizontal_shape(i)) &
            call fatal_error('set_bottom_index', 'shape of provided index array does not match domain extents provided to set_domain.')
      end do
#  endif

      self%domain%bottom_indices => indices
   end subroutine set_bottom_index
#endif

   ! ----------------------------------------------------------------------
   ! start: prepare for simulation start
   ! ----------------------------------------------------------------------
   ! This tells FABM that the user/host have finished providing (or overriding)
   ! data (link_data procedures) and have finished flagging diagnostics for
   ! output (by setting the "save" flag that is part of the variable metadata)
   ! ----------------------------------------------------------------------
   subroutine start(self)
      class (type_fabm_model), intent(inout), target :: self

      integer                                :: ivar
      logical                                :: ready
      type (type_variable_node),pointer      :: variable_node
      type (type_link), pointer              :: link
      character(len=*), parameter            :: log_prefix = 'fabm_'
      integer                                :: log_unit, ios
      class (type_fabm_variable), pointer :: pvariables(:)

      if (self%status < status_set_domain_done) then
         call fatal_error('start', 'set_domain has not yet been called on this model object.')
         return
      elseif (self%status >= status_start_done) then
         ! start has been called on this model before and it must have succeeded to have this status.
         ! Reset store (e.g., all diagnostics) by setting each variable to its fill value and return.
         ! (this allows masked cells to be properly initialized if the mask changes between calls to start)
         call reset_store(self)
         return
      end if

      ready = .true.

#ifdef _HAS_MASK_
#  ifndef _FABM_HORIZONTAL_MASK_
      if (.not. associated(self%domain%mask)) then
         call log_message('spatial mask has not been set. Make sure to call set_mask.')
         ready = .false.
      end if
#  endif
      if (.not. associated(self%domain%mask_hz)) then
         call log_message('horizontal spatial mask has not been set. Make sure to call set_mask.')
         ready = .false.
      end if
#endif

#if defined(_FABM_DEPTH_DIMENSION_INDEX_)&&_FABM_BOTTOM_INDEX_==-1
      if (.not. associated(self%domain%bottom_indices)) then
         call log_message('bottom indices have not been set. Make sure to call set_bottom_index.')
         ready = .false.
      end if
#endif

      ! Flag variables that have had data asssigned (by user, host or FABM).
      ! This is done only now because the user/host had till this moment to provide (or override) model fields.
```

```
747       call flag_variables_with_data(self%variable_register%catalog%interior, self%catalog%interior_sources)
748       call flag_variables_with_data(self%variable_register%catalog%horizontal, self%catalog%horizontal_sources)
749       call flag_variables_with_data(self%variable_register%catalog%scalar, self%catalog%scalar_sources)
750
751       ! Create job that ensures all diagnostics required by the user are computed.
752       ! This is done only now because the user/host had till this moment to change the "save" flag of each diagnostic
.
753       do ivar = 1, size(self%interior_diagnostic_variables)
754          if (self%interior_diagnostic_variables(ivar)%save) then
755             select case (self%interior_diagnostic_variables(ivar)%target%source)
756             case (source_check_state)
757                call self%check_interior_state_job%request_variable(self%interior_diagnostic_variables(ivar)%target, s
tore=.true.)
758             case (source_get_vertical_movement)
759                call self%get_vertical_movement_job%request_variable(self%interior_diagnostic_variables(ivar)%target,
store=.true.)
760             case default
761                call self%finalize_outputs_job%request_variable(self%interior_diagnostic_variables(ivar)%target, store
=.true.)
762             end select
763          end if
764       end do
765       do ivar = 1, size(self%horizontal_diagnostic_variables)
766          if (self%horizontal_diagnostic_variables(ivar)%save) &
767             call self%finalize_outputs_job%request_variable(self%horizontal_diagnostic_variables(ivar)%target, store=
.true.)
768       end do
769
770       log_unit = -1
771       if (self%log) log_unit = get_free_unit()
772
773       ! Merge write indices when operations can be done in place
774       ! This must be done after all variables are requested from the different jobs, so we know which variables
775       ! will be needed separately (such variables cannot be merged)
776       if (self%log) then
777          open(unit=log_unit, file=log_prefix // 'merges.log', action='write', status='replace', iostat=ios)
778          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'merges.log')
779          call merge_indices(self%root, log_unit)
780          close(log_unit)
781       else
782          call merge_indices(self%root)
783       end if
784
785       ! Initialize all jobs. This also creates registers for the read and write caches, as well as the persistent sto
re.
786       if (self%log) then
787          open(unit=log_unit, file=log_prefix // 'task_order.log', action='write', status='replace', iostat=ios)
788          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'task_order.log')
789       end if
790       call self%job_manager%initialize(self%variable_register, self%schedules, log_unit, self%finalize_outputs_job)
791       if (self%log) then
792          close(log_unit)
793          open(unit=log_unit, file=log_prefix // 'graph.gv', action='write', status='replace', iostat=ios)
794          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'graph.gv')
795          call self%job_manager%write_graph(log_unit)
796          close(log_unit)
797       end if
798
799       ! Create persistent store. This provides memory for all variables to be stored there.
800       call create_store(self)
801
802       ! Collect fill values and missing values for cache entries.
803       self%cache_fill_values = get_cache_fill_values(self%variable_register)
804
805       ! Create global caches for exchanging information with BGC models.
806       ! This can only be done after get_cache_fill_values completes, because that determines what values to prefill t
he cache with.
807       call cache_create(self%domain, self%cache_fill_values, self%cache_int)
808       call cache_create(self%domain, self%cache_fill_values, self%cache_hz)
809       call cache_create(self%domain, self%cache_fill_values, self%cache_vert)
810
811       ! For diagnostics that are not needed, set their write index to 0 (rubbish bin)
812       if (self%log) then
813          open(unit=log_unit, file=log_prefix // 'discards.log', action='write', status='replace', iostat=ios)
814          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'discards.log')
815          write (log_unit,'(a)') 'Writes for the following variables are discarded:'
816       end if
817       link => self%links_postcoupling%first
818       do while (associated(link))
819          if (.not. link%target%write_indices%is_empty() .and. link%target%write_indices%value == -1) then
820             call link%target%write_indices%set_value(0)
821             if (self%log) write (log_unit,'("- ",a)') trim(link%target%name)
822          end if
823          link => link%next
824       end do
825       if (self%log) close(log_unit)
826
827       if (self%log) then
828          open(unit=log_unit, file=log_prefix // 'register.log', action='write', status='replace', iostat=ios)
829          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'register.log')
830          call self%variable_register%print(log_unit)
831          close(log_unit)
832
833          open(unit=log_unit, file=log_prefix // 'jobs.log', action='write', status='replace', iostat=ios)
834          if (ios /= 0) call fatal_error('start', 'Unable to open ' // log_prefix // 'jobs.log')
835          call self%job_manager%print(log_unit)
836          close(log_unit)
837       end if
```

```
838
839       ! Report all unfulfilled dependencies.
840       variable_node => self%variable_register%unfulfilled_dependencies%first
841       do while (associated(variable_node))
842          call report_unfulfilled_dependency(variable_node%target)
843          variable_node => variable_node%next
844       end do
845
846       ! Gather information for check_state routines in cache-friendly data structures
847       ! NB intermediate pointer "pvariables" is needed to work around bug in PGI 19.1, 20.7
848       pvariables => self%interior_state_variables
849       call gather_check_state_data(pvariables, self%check_interior_state_data)
850       pvariables => self%surface_state_variables
851       call gather_check_state_data(pvariables, self%check_surface_state_data)
852       pvariables => self%bottom_state_variables
853       call gather_check_state_data(pvariables, self%check_bottom_state_data)
854
855       if (associated(self%variable_register%unfulfilled_dependencies%first) .or. .not. ready) &
856          call fatal_error('start', 'FABM is lacking required data.')
857
858       self%status = status_start_done
859
860    contains
861
862       subroutine gather_check_state_data(variables, dat)
863          class (type_fabm_variable), intent(in)    :: variables(:)
864          type (type_check_state_data), allocatable :: dat(:)
865          allocate(dat(size(variables)))
866          do ivar = 1, size(variables)
867             dat(ivar)%index = variables(ivar)%target%read_indices%value
868             dat(ivar)%minimum = variables(ivar)%target%minimum
869             dat(ivar)%maximum = variables(ivar)%target%maximum
870          end do
871       end subroutine
872
873       subroutine flag_variables_with_data(variable_list, data_sources)
874          type (type_variable_list), intent(inout) :: variable_list
875          integer,                   intent(in)    :: data_sources(:)
876
877          integer :: i
878          type (type_variable_node), pointer :: variable_node
879
880          variable_node => variable_list%first
881          do i = 1, variable_list%count
882             if (data_sources(i) > data_source_fabm) then
883                variable_node%target%source = source_external
884                variable_node%target%write_operator = operator_assign
885             elseif (data_sources(i) /= data_source_none .and. variable_node%target%source == source_unknown) then
886                variable_node%target%source = source_external
887             end if
888             variable_node => variable_node%next
889          end do
890       end subroutine
891
892       subroutine report_unfulfilled_dependency(variable)
893          type (type_internal_variable), target :: variable
894
895          type type_model_reference
896             class (type_base_model),     pointer :: p    => null()
897             type (type_model_reference), pointer :: next => null()
898          end type
899
900          type (type_model_reference), pointer :: first, current, next
901          type (type_link),            pointer :: link
902          logical,                     pointer :: pmember
903          character(len=attribute_length)      :: path
904
905          call log_message('UNFULFILLED DEPENDENCY: ' // trim(variable%name))
906          select case (variable%domain)
907          case (domain_interior)
908             call log_message('  This is an interior field.')
909          case (domain_horizontal, domain_surface, domain_bottom)
910             call log_message('  This is a horizontal field.')
911          case (domain_scalar)
912             call log_message('  This is a scalar field (single value valid across the entire domain).')
913          end select
914          if (variable%units /= '') call log_message('  It has units ' // trim(variable%units))
915          call log_message('  It is needed by the following model instances:')
916
917          first => null()
918          link => self%root%links%first
919          do while (associated(link))
920             if (      associated(link%target, variable)    &                     ! This link points to the target var
    iable,
921                   .and. associated(link%original%read_index) &                     ! the model that owns the link reque
    sts read access for it,
922                   .and. link%original%presence /= presence_external_optional) then ! and this access is required, not o
    ptional
923                current => first
924                pmember => link%original%owner%frozen
925                do while (associated(current))
926                   ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type m
    ember references.
927                   if (associated(pmember, current%p%frozen)) exit
928                   current => current%next
929                end do
930                if (.not. associated(current)) then
931                   ! This model has not been reported before. Do so now and remember that we have done so.
```

```
                  allocate(current)
                  current%p => link%original%owner
                  current%next => first
                  first => current
                  path = current%p%get_path()
                  call log_message('    ' // trim(path(2:)))
               end if
            end if
            link => link%next
         end do

         ! Clean up model list
         current => first
         do while (associated(current))
            next => current%next
            deallocate(current)
            current => next
         end do
      end subroutine
   end subroutine start

   ! ------------------------------------------------------------------------
   ! get_interior_variable_id_by_name: get interior variable identifier for
   ! given variable name
   ! ------------------------------------------------------------------------
   function get_interior_variable_id_by_name(self, name) result(id)
      class (type_fabm_model), intent(in)   :: self
      character(len=*),        intent(in)   :: name
      type (type_fabm_interior_variable_id) :: id

      id%variable => get_variable_by_name(self, name, domain_interior)
   end function get_interior_variable_id_by_name

   ! ------------------------------------------------------------------------
   ! get_interior_variable_id_sn: get interior variable identifier for given
   ! standard variable
   ! ------------------------------------------------------------------------
   function get_interior_variable_id_sn(self, standard_variable) result(id)
      class (type_fabm_model),                intent(in) :: self
      type (type_interior_standard_variable), intent(in) :: standard_variable
      type (type_fabm_interior_variable_id)              :: id

      id%variable => get_variable_by_standard_variable(self, standard_variable%resolve())
   end function get_interior_variable_id_sn

   ! ------------------------------------------------------------------------
   ! get_horizontal_variable_id_by_name: get horizontal variable identifier for
   ! given variable name
   ! ------------------------------------------------------------------------
   function get_horizontal_variable_id_by_name(self, name) result(id)
      class (type_fabm_model), intent(in)     :: self
      character(len=*),        intent(in)     :: name
      type (type_fabm_horizontal_variable_id) :: id

      id%variable => get_variable_by_name(self, name, domain_horizontal)
   end function get_horizontal_variable_id_by_name

   ! ------------------------------------------------------------------------
   ! get_horizontal_variable_id_sn: get horizontal variable identifier for
   ! given standard variable
   ! ------------------------------------------------------------------------
   function get_horizontal_variable_id_sn(self, standard_variable) result(id)
      class (type_fabm_model),                  intent(in) :: self
      class (type_horizontal_standard_variable), intent(in) :: standard_variable
      type (type_fabm_horizontal_variable_id)              :: id

      id%variable => get_variable_by_standard_variable(self, standard_variable%resolve())
   end function get_horizontal_variable_id_sn

   ! ------------------------------------------------------------------------
   ! get_scalar_variable_id_by_name: get scalar variable identifier for given
   ! variable name
   ! ------------------------------------------------------------------------
   function get_scalar_variable_id_by_name(self, name) result(id)
      class (type_fabm_model), intent(in) :: self
      character(len=*),        intent(in) :: name
      type (type_fabm_scalar_variable_id) :: id

      id%variable => get_variable_by_name(self, name, domain_scalar)
   end function get_scalar_variable_id_by_name

   ! ------------------------------------------------------------------------
   ! get_scalar_variable_id_sn: get scalar variable identifier for given
   ! standard variable
   ! ------------------------------------------------------------------------
   function get_scalar_variable_id_sn(self, standard_variable) result(id)
      class (type_fabm_model),              intent(in) :: self
      type (type_global_standard_variable), intent(in) :: standard_variable
      type (type_fabm_scalar_variable_id)              :: id

      id%variable => get_variable_by_standard_variable(self, standard_variable%resolve())
   end function get_scalar_variable_id_sn

   ! ------------------------------------------------------------------------
   ! get_variable_name: get output name associated with given variable id.
   ! The name consists of alphanumeric characters and underscores only.
   ! ------------------------------------------------------------------------
   function get_variable_name(self, id) result(name)
```

```
      class (type_fabm_model),       intent(in) :: self
      class (type_fabm_variable_id), intent(in) :: id
      character(len=attribute_length)           :: name

      name = ''
      if (associated(id%variable)) name = get_safe_name(id%variable%name)
   end function get_variable_name

   ! ------------------------------------------------------------------------
   ! is_variable_used: returns whether this variable is an input for any
   ! biogeochemical module
   ! ------------------------------------------------------------------------
   function is_variable_used(id) result(used)
      class (type_fabm_variable_id), intent(in) :: id
      logical                                   :: used

      used = associated(id%variable)
      if (used) used = .not. id%variable%read_indices%is_empty()
   end function is_variable_used

   ! ------------------------------------------------------------------------
   ! interior_variable_needs_values: returns whether values still need to
   ! provided for this interior variable, identified by id.
   ! Unless these values are provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function interior_variable_needs_values(self, id) result(required)
      class (type_fabm_model),               intent(in) :: self
      type (type_fabm_interior_variable_id), intent(in) :: id
      logical                                          :: required

      required = associated(id%variable)
      if (required) required = .not. id%variable%read_indices%is_empty()
      if (required) required = .not. associated(self%catalog%interior(id%variable%catalog_index)%p)
   end function interior_variable_needs_values

   ! ------------------------------------------------------------------------
   ! interior_variable_needs_values_sn: returns whether values still need to
   ! provided for this interior variable, identified by standard variable.
   ! Unless these values are provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function interior_variable_needs_values_sn(self, standard_variable) result(required)
      class (type_fabm_model),                intent(in) :: self
      type (type_interior_standard_variable), intent(in) :: standard_variable
      logical                                            :: required

      required = interior_variable_needs_values(self, get_interior_variable_id_sn(self, standard_variable))
   end function interior_variable_needs_values_sn

   ! ------------------------------------------------------------------------
   ! horizontal_variable_needs_values: returns whether values still need to
   ! provided for this horizontal variable, identified by id.
   ! Unless these values are provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function horizontal_variable_needs_values(self, id) result(required)
      class (type_fabm_model),                 intent(in) :: self
      type (type_fabm_horizontal_variable_id), intent(in) :: id
      logical                                            :: required

      required = associated(id%variable)
      if (required) required = .not. id%variable%read_indices%is_empty()
      if (required) required = .not. associated(self%catalog%horizontal(id%variable%catalog_index)%p)
   end function horizontal_variable_needs_values

   ! ------------------------------------------------------------------------
   ! horizontal_variable_needs_values_sn: returns whether values still need to
   ! provided for this horizontal variable, identified by standard variable.
   ! Unless these values are provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function horizontal_variable_needs_values_sn(self, standard_variable) result(required)
      class (type_fabm_model),                   intent(in) :: self
      class (type_horizontal_standard_variable), intent(in) :: standard_variable
      logical                                              :: required

      required = horizontal_variable_needs_values(self, get_horizontal_variable_id_sn(self, standard_variable))
   end function horizontal_variable_needs_values_sn

   ! ------------------------------------------------------------------------
   ! scalar_variable_needs_values: returns whether a value still need to
   ! provided for this scalar variable, identified by id.
   ! Unless this value is provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function scalar_variable_needs_values(self, id) result(required)
      class (type_fabm_model),              intent(in) :: self
      type (type_fabm_scalar_variable_id), intent(in) :: id
      logical                                         :: required

      required = associated(id%variable)
      if (required) required = .not. id%variable%read_indices%is_empty()
      if (required) required = .not. associated(self%catalog%scalar(id%variable%catalog_index)%p)
   end function scalar_variable_needs_values

   ! ------------------------------------------------------------------------
   ! scalar_variable_needs_values: returns whether a value still need to
   ! provided for this scalar variable, identified by standard variable.
   ! Unless this value is provided, a call to "start" will fail.
   ! ------------------------------------------------------------------------
   function scalar_variable_needs_values_sn(self, standard_variable) result(required)
      class (type_fabm_model),               intent(in) :: self
```

```
      type (type_global_standard_variable), intent(in) :: standard_variable
      logical                                          :: required

      required = scalar_variable_needs_values(self, get_scalar_variable_id_sn(self, standard_variable))
   end function scalar_variable_needs_values_sn

   subroutine require_interior_data(self, standard_variable)
      class (type_fabm_model),                intent(inout) :: self
      type (type_interior_standard_variable), intent(in)    :: standard_variable

      type (type_fabm_interior_variable_id) :: id

      if (self%status < status_initialize_done) &
         call fatal_error('require_interior_data', 'This procedure can only be called after model initialization.')
      if (self%status >= status_start_done) &
         call fatal_error('require_interior_data', 'This procedure cannot be called after start is called.')

      id = self%get_interior_variable_id(standard_variable)
      if (.not. associated(id%variable)) &
         call fatal_error('require_interior_data', 'Model does not contain requested variable ' // trim(standard_vari
able%name))
      call self%finalize_outputs_job%request_variable(id%variable, store=.true.)
   end subroutine require_interior_data

   subroutine require_horizontal_data(self, standard_variable)
      class (type_fabm_model),                  intent(inout) :: self
      class (type_horizontal_standard_variable), intent(in)    :: standard_variable

      type (type_fabm_horizontal_variable_id) :: id

      if (self%status < status_initialize_done) &
         call fatal_error('require_horizontal_data', 'This procedure can only be called after model initialization.')

      if (self%status >= status_start_done) &
         call fatal_error('require_horizontal_data', 'This procedure cannot be called after check_ready is called.')

      id = self%get_horizontal_variable_id(standard_variable)
      if (.not. associated(id%variable)) &
         call fatal_error('require_horizontal_data', 'Model does not contain requested variable ' // trim(standard_va
riable%name))
      call self%finalize_outputs_job%request_variable(id%variable, store=.true.)
   end subroutine require_horizontal_data

   subroutine link_interior_data_by_variable(self, variable, dat, source)
      class (type_fabm_model),               intent(inout) :: self
      type (type_internal_variable),         intent(in)    :: variable
      real(rke) _ATTRIBUTES_GLOBAL_, target, intent(in)    :: dat
      integer, optional,                     intent(in)    :: source

      integer :: i
      integer :: source_

#if !defined(NĐEBUG)&&_FABM_ĐIMENSION_COUNT_>0
      do i = 1, size(self%domain%shape)
         if (size(dat, i) /= self%domain%shape(i)) then
            call fatal_error('link_interior_data_by_variable', trim(variable%name) // &
               ': extents of provided array do not match domain extents.')
         end if
      end do
#endif
      _ASSERT_(variable%domain == domain_interior, 'link_interior_data_by_variable', 'link_interior_data_by_variable
called with variable without domain_interior.')

      i = variable%catalog_index
      if (i /= -1) then
         source_ = data_source_default
         if (present(source)) source_ = source
         if (source_ >= self%catalog%interior_sources(i)) then
            self%catalog%interior(i)%p => dat
            self%catalog%interior_sources(i) = source_
         end if
      end if
   end subroutine link_interior_data_by_variable

   subroutine link_interior_data_by_id(self, id, dat, source)
      class (type_fabm_model),               intent(inout) :: self
      type(type_fabm_interior_variable_id),  intent(in)    :: id
      real(rke) _ATTRIBUTES_GLOBAL_, target, intent(in)    :: dat
      integer,optional,                      intent(in)    :: source

      if (associated(id%variable)) call link_interior_data_by_variable(self, id%variable, dat, source)
   end subroutine link_interior_data_by_id

   subroutine link_interior_data_by_sn(self, standard_variable, dat)
      class (type_fabm_model),                intent(inout) :: self
      type (type_interior_standard_variable), intent(in)    :: standard_variable
      real(rke) _ATTRIBUTES_GLOBAL_, target,  intent(in)    :: dat

      call link_interior_data_by_id(self, get_interior_variable_id_sn(self, standard_variable), dat)
   end subroutine link_interior_data_by_sn

   subroutine link_interior_data_by_name(self, name, dat)
      class (type_fabm_model),       target, intent(inout) :: self
      character(len=*),                      intent(in)    :: name
      real(rke) _ATTRIBUTES_GLOBAL_, target, intent(in)    :: dat

      call link_interior_data_by_id(self, get_interior_variable_id_by_name(self, name), dat)
   end subroutine link_interior_data_by_name
```

```
1222
1223    subroutine link_horizontal_data_by_variable(self, variable, dat, source)
1224       class (type_fabm_model),                          intent(inout) :: self
1225       type (type_internal_variable),                    intent(in)    :: variable
1226       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1227       integer, optional,                                intent(in)    :: source
1228
1229       integer :: i
1230       integer :: source_
1231
1232 #if !defined(NDEBUG)&&_HORIZONTAL_DIMENSION_COUNT_>0
1233       do i = 1, size(self%domain%horizontal_shape)
1234          if (size(dat, i) /= self%domain%horizontal_shape(i)) then
1235             call fatal_error('link_horizontal_data_by_variable', trim(variable%name) // &
1236                ': extents of provided array do not match domain extents.')
1237          end if
1238       end do
1239 #endif
1240       _ASSERT_(iand(variable%domain, domain_horizontal) /= 0, 'link_horizontal_data_by_variable', 'link_horizontal_da
      ta_by_variable called with variable without domain_horizontal.')
1241
1242       i = variable%catalog_index
1243       if (i /= -1) then
1244          source_ = data_source_default
1245          if (present(source)) source_ = source
1246          if (source_ >= self%catalog%horizontal_sources(i)) then
1247             self%catalog%horizontal(i)%p => dat
1248             self%catalog%horizontal_sources(i) = source_
1249          end if
1250       end if
1251    end subroutine link_horizontal_data_by_variable
1252
1253    subroutine link_horizontal_data_by_id(self, id, dat, source)
1254       class (type_fabm_model),                          intent(inout) :: self
1255       type (type_fabm_horizontal_variable_id),          intent(in)    :: id
1256       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1257       integer, optional,                                intent(in)    :: source
1258
1259       if (associated(id%variable)) call link_horizontal_data_by_variable(self, id%variable, dat, source)
1260    end subroutine link_horizontal_data_by_id
1261
1262    subroutine link_horizontal_data_by_sn(self, standard_variable, dat)
1263       class (type_fabm_model),                          intent(inout) :: self
1264       class (type_horizontal_standard_variable),        intent(in)    :: standard_variable
1265       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1266
1267       call link_horizontal_data_by_id(self, get_horizontal_variable_id_sn(self, standard_variable), dat)
1268    end subroutine link_horizontal_data_by_sn
1269
1270    subroutine link_horizontal_data_by_name(self, name, dat)
1271       class (type_fabm_model),                          intent(inout) :: self
1272       character(len=*),                                 intent(in)    :: name
1273       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1274
1275       call link_horizontal_data_by_id(self, get_horizontal_variable_id_by_name(self, name), dat)
1276    end subroutine link_horizontal_data_by_name
1277
1278    subroutine link_scalar_by_variable(self, variable, dat, source)
1279       class (type_fabm_model),             intent(inout) :: self
1280       type (type_internal_variable),       intent(in)    :: variable
1281       real(rke), target,                   intent(in)    :: dat
1282       integer, optional,                   intent(in)    :: source
1283
1284       integer :: i
1285       integer :: source_
1286
1287       _ASSERT_(variable%domain == domain_scalar, 'link_scalar_by_variable', 'link_scalar_by_variable called with vari
      able without domain_scalar.')
1288       i = variable%catalog_index
1289       if (i /= -1) then
1290          source_ = data_source_default
1291          if (present(source)) source_ = source
1292          if (source_ >= self%catalog%scalar_sources(i)) then
1293             self%catalog%scalar(i)%p => dat
1294             self%catalog%scalar_sources(i) = source_
1295          end if
1296       end if
1297    end subroutine link_scalar_by_variable
1298
1299    subroutine link_scalar_by_id(self, id, dat, source)
1300       class (type_fabm_model),             intent(inout) :: self
1301       type (type_fabm_scalar_variable_id), intent(in)    :: id
1302       real(rke), target,                   intent(in)    :: dat
1303       integer, optional,                   intent(in)    :: source
1304
1305       if (associated(id%variable)) call link_scalar_by_variable(self, id%variable, dat, source)
1306    end subroutine link_scalar_by_id
1307
1308    subroutine link_scalar_by_sn(self, standard_variable, dat)
1309       class (type_fabm_model),             intent(inout) :: self
1310       type(type_global_standard_variable), intent(in)    :: standard_variable
1311       real(rke), target,                   intent(in)    :: dat
1312
1313       call link_scalar_by_id(self, get_scalar_variable_id_sn(self, standard_variable), dat)
1314    end subroutine link_scalar_by_sn
1315
1316    subroutine link_scalar_by_name(self, name, dat)
1317       class (type_fabm_model), intent(inout) :: self
```

```
1318       character(len=*),        intent(in)    :: name
1319       real(rke), target,       intent(in)    :: dat
1320
1321       call link_scalar_by_id(self, get_scalar_variable_id_by_name(self, name), dat)
1322    end subroutine link_scalar_by_name
1323
1324    subroutine link_interior_state_data(self, index, dat)
1325       class (type_fabm_model),               intent(inout) :: self
1326       integer,                               intent(in)    :: index
1327       real(rke) _ATTRIBUTES_GLOBAL_, target, intent(in)    :: dat
1328
1329       call link_interior_data_by_variable(self, self%interior_state_variables(index)%target, dat, source=data_source_
fabm)
1330    end subroutine link_interior_state_data
1331
1332    subroutine link_bottom_state_data(self, index, dat)
1333       class (type_fabm_model),                           intent(inout) :: self
1334       integer,                                           intent(in)    :: index
1335       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1336
1337       call link_horizontal_data_by_variable(self, self%bottom_state_variables(index)%target, dat, source=data_source_
fabm)
1338    end subroutine link_bottom_state_data
1339
1340    subroutine link_surface_state_data(self, index, dat)
1341       class (type_fabm_model),                           intent(inout) :: self
1342       integer,                                           intent(in)    :: index
1343       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, target, intent(in)    :: dat
1344
1345       call link_horizontal_data_by_variable(self, self%surface_state_variables(index)%target, dat, source=data_source
_fabm)
1346    end subroutine link_surface_state_data
1347
1348    subroutine link_all_interior_state_data(self, dat)
1349       class (type_fabm_model),                    intent(inout) :: self
1350       real(rke) _DIMENSION_GLOBAL_PLUS_1_, target, intent(in)    :: dat
1351
1352       integer                                  :: i
1353       real(rke) _ATTRIBUTES_GLOBAL_, pointer :: pdat
1354
1355 #ifndef NDEBUG
1356       if (size(dat, _FABM_DIMENSION_COUNT_ + 1) /= size(self%interior_state_variables)) &
1357          call fatal_error('link_all_interior_state_data', 'size of last dimension of provided array must match number
 of interior state variables.')
1358 #endif
1359       do i = 1, size(self%interior_state_variables)
1360          pdat => dat(_PREARG_LOCATION_DIMENSIONS_ i)
1361          call link_interior_state_data(self, i, pdat)
1362       end do
1363    end subroutine link_all_interior_state_data
1364
1365    subroutine link_all_bottom_state_data(self, dat)
1366       class (type_fabm_model),                                intent(inout) :: self
1367       real(rke) _DIMENSION_GLOBAL_HORIZONTAL_PLUS_1_, target, intent(in)    :: dat
1368
1369       integer                                             :: i
1370       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, pointer :: pdat
1371
1372 #ifndef NDEBUG
1373       if (size(dat, _HORIZONTAL_DIMENSION_COUNT_ + 1) /= size(self%bottom_state_variables)) &
1374          call fatal_error('link_all_bottom_state_data', 'size of last dimension of provided array must match number o
f bottom state variables.')
1375 #endif
1376       do i = 1, size(self%bottom_state_variables)
1377          pdat => dat(_PREARG_HORIZONTAL_LOCATION_DIMENSIONS_ i)
1378          call link_bottom_state_data(self, i, pdat)
1379       end do
1380    end subroutine link_all_bottom_state_data
1381
1382    subroutine link_all_surface_state_data(self, dat)
1383       class (type_fabm_model),                                intent(inout) :: self
1384       real(rke) _DIMENSION_GLOBAL_HORIZONTAL_PLUS_1_, target, intent(in)    :: dat
1385
1386       integer                                             :: i
1387       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, pointer :: pdat
1388
1389 #ifndef NDEBUG
1390       if (size(dat, _HORIZONTAL_DIMENSION_COUNT_ + 1) /= size(self%surface_state_variables)) &
1391          call fatal_error('link_all_surface_state_data', 'size of last dimension of provided array must match number
of surface state variables.')
1392 #endif
1393       do i = 1, size(self%surface_state_variables)
1394          pdat => dat(_PREARG_HORIZONTAL_LOCATION_DIMENSIONS_ i)
1395          call link_surface_state_data(self, i, pdat)
1396       end do
1397    end subroutine link_all_surface_state_data
1398
1399    function get_interior_diagnostic_data(self, index) result(dat)
1400       class (type_fabm_model), intent(in)    :: self
1401       integer,                 intent(in)    :: index
1402       real(rke) _ATTRIBUTES_GLOBAL_, pointer :: dat
1403
1404       _ASSERT_(self%status >= status_start_done, 'get_interior_diagnostic_data', 'This routine can only be called aft
er model start.')
1405       dat => null()
1406       if (self%interior_diagnostic_variables(index)%target%catalog_index /= -1) &
1407          dat => self%catalog%interior(self%interior_diagnostic_variables(index)%target%catalog_index)%p
1408    end function get_interior_diagnostic_data
```

```
1409
1410    function get_horizontal_diagnostic_data(self, index) result(dat)
1411       class (type_fabm_model), intent(in)               :: self
1412       integer,                 intent(in)               :: index
1413       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, pointer :: dat
1414
1415       _ASSERT_(self%status >= status_start_done, 'get_horizontal_diagnostic_data', 'This routine can only be called a
     fter model start.')
1416       dat => null()
1417       if (self%horizontal_diagnostic_variables(index)%target%catalog_index /= -1) &
1418          dat => self%catalog%horizontal(self%horizontal_diagnostic_variables(index)%target%catalog_index)%p
1419    end function get_horizontal_diagnostic_data
1420
1421    function get_interior_data(self, id) result(dat)
1422       class (type_fabm_model), target,       intent(in) :: self
1423       type (type_fabm_interior_variable_id), intent(in) :: id
1424       real(rke) _ATTRIBUTES_GLOBAL_, pointer            :: dat
1425
1426       _ASSERT_(self%status >= status_start_done, 'get_interior_data', 'This routine can only be called after model st
     art.')
1427       dat => null()
1428       if (.not. associated(id%variable)) return
1429       if (id%variable%catalog_index /= -1) dat => self%catalog%interior(id%variable%catalog_index)%p
1430    end function get_interior_data
1431
1432    function get_horizontal_data(self, id) result(dat)
1433       class (type_fabm_model), target,        intent(in) :: self
1434       type(type_fabm_horizontal_variable_id), intent(in) :: id
1435       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, pointer  :: dat
1436
1437       _ASSERT_(self%status >= status_start_done, 'get_horizontal_data', 'This routine can only be called after model
     start.')
1438       dat => null()
1439       if (.not. associated(id%variable)) return
1440       if (id%variable%catalog_index /= -1) dat => self%catalog%horizontal(id%variable%catalog_index)%p
1441    end function get_horizontal_data
1442
1443    function get_scalar_data(self, id) result(dat)
1444       class (type_fabm_model), target,     intent(in) :: self
1445       type (type_fabm_scalar_variable_id), intent(in) :: id
1446       real(rke), pointer                              :: dat
1447
1448       _ASSERT_(self%status >= status_start_done, 'get_scalar_data', 'This routine can only be called after model star
     t.')
1449       dat => null()
1450       if (.not. associated(id%variable)) return
1451       if (id%variable%catalog_index /= -1) dat => self%catalog%scalar(id%variable%catalog_index)%p
1452    end function get_scalar_data
1453
1454    subroutine initialize_interior_state(self _POSTARG_INTERIOR_IN_)
1455       class (type_fabm_model), intent(inout) :: self
1456       _DECLARE_ARGUMENTS_INTERIOR_IN_
1457
1458       integer :: ivar, read_index, icall
1459       _DECLARE_INTERIOR_INDICES_
1460
1461 #ifndef NDEBUG
1462       call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'initialize_interior_st
     ate')
1463 #endif
1464
1465       call cache_pack(self%domain, self%catalog, self%cache_fill_values, self%initialize_interior_state_job%first_tas
     k, self%cache_int _POSTARG_INTERIOR_IN_)
1466
1467       ! Default initialization for interior state variables
1468       do ivar = 1, size(self%interior_state_variables)
1469          read_index = self%interior_state_variables(ivar)%target%read_indices%value
1470          _CONCURRENT_LOOP_BEGIN_EX_(self%cache_int)
1471             self%cache_int%read _INDEX_SLICE_PLUS_1_(read_index) = self%interior_state_variables(ivar)%initial_value
1472          _LOOP_END_
1473       end do
1474
1475       ! Allow biogeochemical models to initialize their interior state.
1476       do icall = 1, size(self%initialize_interior_state_job%first_task%calls)
1477          if (self%initialize_interior_state_job%first_task%calls(icall)%source == source_initialize_state) call self%
     initialize_interior_state_job%first_task%calls(icall)%model%initialize_state(self%cache_int)
1478       end do
1479
1480       ! Copy from cache back to global data store [NB variable values have been set in the *read* cache].
1481       do ivar = 1, size(self%interior_state_variables)
1482          read_index = self%interior_state_variables(ivar)%target%read_indices%value
1483          if (self%catalog%interior_sources(read_index) == data_source_fabm) then
1484             _UNPACK_TO_GLOBAL_(self%cache_int%read, read_index, self%catalog%interior(self%interior_state_variables(i
     var)%target%catalog_index)%p, self%cache_int, self%interior_state_variables(ivar)%missing_value)
1485          end if
1486       end do
1487
1488       call cache_unpack(self%initialize_interior_state_job%first_task, self%cache_int, self%store _POSTARG_INTERIOR_I
     N_)
1489    end subroutine initialize_interior_state
1490
1491    subroutine initialize_bottom_state(self _POSTARG_HORIZONTAL_IN_)
1492       class (type_fabm_model), intent(inout) :: self
1493       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1494
1495       integer :: icall, ivar, read_index
1496       _DECLARE_HORIZONTAL_INDICES_
1497
```

```
1498 #ifndef NDEBUG
1499       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'initialize_bottom_
     state')
1500 #endif
1501
1502       call cache_pack(self%domain, self%catalog, self%cache_fill_values, self%initialize_bottom_state_job%first_task,
      self%cache_hz _POSTARG_HORIZONTAL_IN_)
1503
1504       ! Default initialization for bottom state variables
1505       do ivar = 1, size(self%bottom_state_variables)
1506          read_index = self%bottom_state_variables(ivar)%target%read_indices%value
1507          _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
1508             self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index) = self%bottom_state_variables(ivar)%ini
     tial_value
1509          _HORIZONTAL_LOOP_END_
1510       end do
1511
1512       ! Allow biogeochemical models to initialize their bottom state.
1513       do icall = 1, size(self%initialize_bottom_state_job%first_task%calls)
1514          if (self%initialize_bottom_state_job%first_task%calls(icall)%source == source_initialize_bottom_state) call
     self%initialize_bottom_state_job%first_task%calls(icall)%model%initialize_bottom_state(self%cache_hz)
1515       end do
1516
1517       ! Copy from cache back to global data store [NB variable values have been set in the *read* cache].
1518       do ivar = 1, size(self%bottom_state_variables)
1519          read_index = self%bottom_state_variables(ivar)%target%read_indices%value
1520          if (self%catalog%horizontal_sources(read_index) == data_source_fabm) then
1521             _HORIZONTAL_UNPACK_TO_GLOBAL_(self%cache_hz%read_hz, read_index, self%catalog%horizontal(self%bottom_stat
     e_variables(ivar)%target%catalog_index)%p, self%cache_hz, self%bottom_state_variables(ivar)%missing_value)
1522          end if
1523       end do
1524
1525       call cache_unpack(self%initialize_bottom_state_job%first_task, self%cache_hz, self%store _POSTARG_HORIZONTAL_IN
     _)
1526    end subroutine initialize_bottom_state
1527
1528    subroutine initialize_surface_state(self _POSTARG_HORIZONTAL_IN_)
1529       class (type_fabm_model), intent(inout) :: self
1530       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1531
1532       integer :: icall, ivar, read_index
1533       _DECLARE_HORIZONTAL_INDICES_
1534
1535 #ifndef NDEBUG
1536       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'initialize_surface
     _state')
1537 #endif
1538
1539       call cache_pack(self%domain, self%catalog, self%cache_fill_values, self%initialize_surface_state_job%first_task
     , self%cache_hz _POSTARG_HORIZONTAL_IN_)
1540
1541       ! Default initialization for surface state variables
1542       do ivar = 1, size(self%surface_state_variables)
1543          read_index = self%surface_state_variables(ivar)%target%read_indices%value
1544          _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
1545             self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index) = self%surface_state_variables(ivar)%in
     itial_value
1546          _HORIZONTAL_LOOP_END_
1547       end do
1548
1549       ! Allow biogeochemical models to initialize their surface state.
1550       do icall = 1, size(self%initialize_surface_state_job%first_task%calls)
1551          if (self%initialize_surface_state_job%first_task%calls(icall)%source == source_initialize_surface_state) cal
     l self%initialize_surface_state_job%first_task%calls(icall)%model%initialize_surface_state(self%cache_hz)
1552       end do
1553
1554       ! Copy from cache back to global data store [NB variable values have been set in the *read* cache].
1555       do ivar = 1, size(self%surface_state_variables)
1556          read_index = self%surface_state_variables(ivar)%target%read_indices%value
1557          if (self%catalog%horizontal_sources(read_index) == data_source_fabm) then
1558             _HORIZONTAL_UNPACK_TO_GLOBAL_(self%cache_hz%read_hz, read_index, self%catalog%horizontal(self%surface_sta
     te_variables(ivar)%target%catalog_index)%p, self%cache_hz, self%surface_state_variables(ivar)%missing_value)
1559          end if
1560       end do
1561
1562       call cache_unpack(self%initialize_surface_state_job%first_task, self%cache_hz, self%store _POSTARG_HORIZONTAL_I
     N_)
1563    end subroutine initialize_surface_state
1564
1565    subroutine get_interior_sources_rhs(self _POSTARG_INTERIOR_IN_, dy)
1566       class (type_fabm_model),                intent(inout) :: self
1567       _DECLARE_ARGUMENTS_INTERIOR_IN_
1568       real(rke) _DIMENSION_EXT_SLICE_PLUS_1_, intent(inout) :: dy
1569
1570       integer :: i, k
1571       _DECLARE_INTERIOR_INDICES_
1572
1573 #ifndef NDEBUG
1574       call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'get_interior_sources_r
     hs')
1575 #  ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
1576       call check_extents_2d(dy, _STOP_ - _START_ + 1, size(self%interior_state_variables), 'get_interior_sources_rhs'
     , 'dy', 'stop-start+1, # interior state variables')
1577 #  else
1578       call check_extents_1d(dy, size(self%interior_state_variables), 'get_interior_sources_rhs', 'dy', '# interior st
     ate variables')
1579 #  endif
1580 #endif
```

```
1581
1582        call process_interior_slice(self%get_interior_sources_job%first_task, self%domain, self%catalog, self%cache_fil
    l_values, self%store, self%cache_int _POSTARG_INTERIOR_IN_)
1583
1584        ! Compose total sources-sinks for each state variable, combining model-specific contributions.
1585        do i = 1, size(self%get_interior_sources_job%arg1_sources)
1586           k = self%get_interior_sources_job%arg1_sources(i)
1587           _UNPACK_AND_ADD_TO_PLUS_1_(self%cache_int%write, k, dy, i, self%cache_int)
1588        end do
1589     end subroutine get_interior_sources_rhs
1590
1591     subroutine get_interior_sources_ppdd(self _POSTARG_INTERIOR_IN_, pp, dd)
1592        class (type_fabm_model),                intent(inout) :: self
1593       _DECLARE_ARGUMENTS_INTERIOR_IN_
1594        real(rke) _DIMENSION_EXT_SLICE_PLUS_2_, intent(inout) :: pp, dd
1595
1596        integer :: icall, i, j, k, ncopy
1597        _DECLARE_INTERIOR_INDICES_
1598
1599 #ifndef NDEBUG
1600        call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'get_interior_sources_p
    pdd')
1601 #  ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
1602        call check_extents_3d(pp, _STOP_ - _START_ + 1, size(self%interior_state_variables), size(self%interior_state_v
    ariables), 'get_interior_sources_ppdd', 'pp', 'stop-start+1, # interior state variables, # interior state variables')

1603        call check_extents_3d(dd, _STOP_ - _START_ + 1, size(self%interior_state_variables), size(self%interior_state_v
    ariables), 'get_interior_sources_ppdd', 'dd', 'stop-start+1, # interior state variables, # interior state variables')

1604 #  else
1605        call check_extents_2d(pp, size(self%interior_state_variables), size(self%interior_state_variables), 'get_interi
    or_sources_ppdd', 'pp', '# interior state variables, # interior state variables')
1606        call check_extents_2d(dd, size(self%interior_state_variables), size(self%interior_state_variables), 'get_interi
    or_sources_ppdd', 'dd', '# interior state variables, # interior state variables')
1607 #  endif
1608 #endif
1609
1610        call cache_pack(self%domain, self%catalog, self%cache_fill_values, self%get_interior_sources_job%first_task, se
    lf%cache_int _POSTARG_INTERIOR_IN_)
1611
1612        ncopy = 0
1613        do icall = 1, size(self%get_interior_sources_job%first_task%calls)
1614           call self%get_interior_sources_job%first_task%calls(icall)%model%do_ppdd(self%cache_int, pp, dd)
1615
1616           ! Copy outputs of interest to read cache so consecutive models can use it.
1617           _DO_CONCURRENT_(i,1 + ncopy,self%get_interior_sources_job%first_task%calls(icall)%ncopy_int + ncopy)
1618              j = self%get_interior_sources_job%first_task%copy_commands_int(i)%read_index
1619              k = self%get_interior_sources_job%first_task%copy_commands_int(i)%write_index
1620              _CONCURRENT_LOOP_BEGIN_EX_(self%cache_int)
1621                 self%cache_int%read _INDEX_SLICE_PLUS_1_(j) = self%cache_int%write _INDEX_SLICE_PLUS_1_(k)
1622              _LOOP_END_
1623           end do
1624           ncopy = ncopy + self%get_interior_sources_job%first_task%calls(icall)%ncopy_int
1625        end do
1626
1627        call cache_unpack(self%get_interior_sources_job%first_task, self%cache_int, self%store _POSTARG_INTERIOR_IN_)
1628     end subroutine get_interior_sources_ppdd
1629
1630     subroutine check_interior_state(self _POSTARG_INTERIOR_IN_, repair, valid)
1631        class (type_fabm_model), intent(inout) :: self
1632        _DECLARE_ARGUMENTS_INTERIOR_IN_
1633        logical,                 intent(in)    :: repair
1634        logical,                 intent(out)   :: valid
1635
1636        logical            :: valid_ranges
1637        integer            :: ivar, read_index
1638        real(rki)          :: value, minimum, maximum
1639        character(len=256) :: err
1640        _DECLARE_INTERIOR_INDICES_
1641
1642 #ifndef NDEBUG
1643        call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'check_interior_state')

1644 #endif
1645
1646        self%cache_int%repair = repair
1647        self%cache_int%valid = .true.
1648        self%cache_int%set_interior = .false.
1649
1650        call process_interior_slice(self%check_interior_state_job%first_task, self%domain, self%catalog, self%cache_fil
    l_values, self%store, self%cache_int _POSTARG_INTERIOR_IN_)
1651
1652        valid = self%cache_int%valid
1653        if (.not. (valid .or. repair)) return
1654
1655        ! Finally check whether all state variable values lie within their prescribed [constant] bounds.
1656        ! This is always done, independently of any model-specific checks that may have been called above.
1657
1658        ! Quick bounds check for the common case where all values are valid.
1659        valid_ranges = .true.
1660        do ivar = 1, size(self%check_interior_state_data)
1661           read_index = self%check_interior_state_data(ivar)%index
1662           minimum = self%check_interior_state_data(ivar)%minimum
1663           maximum = self%check_interior_state_data(ivar)%maximum
1664           _LOOP_BEGIN_EX_(self%cache_int)
1665              value = self%cache_int%read _INDEX_SLICE_PLUS_1_(read_index)
1666              if (value < minimum .or. value > maximum) valid_ranges = .false.
1667           _LOOP_END_
```

```
1668       end do
1669       valid = valid .and. valid_ranges
1670
1671       if (.not. valid_ranges) then
1672          ! One or more variables have out-of-bounds value(s)
1673          ! Either repair these by clipping or stop with an error message.
1674          do ivar = 1, size(self%interior_state_variables)
1675             read_index = self%check_interior_state_data(ivar)%index
1676             minimum = self%check_interior_state_data(ivar)%minimum
1677             maximum = self%check_interior_state_data(ivar)%maximum
1678
1679             if (repair) then
1680                _CONCURRENT_LOOP_BEGIN_EX_(self%cache_int)
1681                   value = self%cache_int%read _INDEX_SLICE_PLUS_1_(read_index)
1682                   self%cache_int%read _INDEX_SLICE_PLUS_1_(read_index) = max(minimum, min(maximum, value))
1683                _LOOP_END_
1684             else
1685                _LOOP_BEGIN_EX_(self%cache_int)
1686                   value = self%cache_int%read _INDEX_SLICE_PLUS_1_(read_index)
1687                   if (value < minimum) then
1688                      ! State variable value lies below prescribed minimum.
1689                      write (unit=err,fmt='(a,e12.4,a,a,a,e12.4)') 'Value ',value,' of variable ',trim(self%interior_s
     tate_variables(ivar)%name), &
1690                                                                   & ' below minimum value ',minimum
1691                      call log_message(err)
1692                      return
1693                   elseif (value > maximum) then
1694                      ! State variable value exceeds prescribed maximum.
1695                      write (unit=err,fmt='(a,e12.4,a,a,a,e12.4)') 'Value ',value,' of variable ',trim(self%interior_s
     tate_variables(ivar)%name), &
1696                                                                   & ' above maximum value ',maximum
1697                      call log_message(err)
1698                      return
1699                   end if
1700                _LOOP_END_
1701             end if
1702          end do
1703       end if
1704
1705       if (self%cache_int%set_interior .or. .not. valid_ranges) then
1706          do ivar = 1, size(self%interior_state_variables)
1707             read_index = self%check_interior_state_data(ivar)%index
1708             if (self%catalog%interior_sources(read_index) == data_source_fabm) then
1709                _UNPACK_TO_GLOBAL_(self%cache_int%read, read_index, self%catalog%interior(self%interior_state_variable
     s(ivar)%target%catalog_index)%p, self%cache_int, self%interior_state_variables(ivar)%missing_value)
1710             end if
1711          end do
1712       end if
1713    end subroutine check_interior_state
1714
1715    subroutine check_bottom_state(self _POSTARG_HORIZONTAL_IN_, repair, valid)
1716       class (type_fabm_model), intent(inout) :: self
1717       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1718       logical,                 intent(in)    :: repair
1719       logical,                 intent(out)   :: valid
1720
1721 #ifndef NDEBUG
1722       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'check_bottom_state
     ')
1723 #endif
1724
1725       call internal_check_horizontal_state(self, self%check_bottom_state_job _POSTARG_HORIZONTAL_IN_, self%check_bott
     om_state_data, 2, self%bottom_state_variables, repair, valid)
1726    end subroutine check_bottom_state
1727
1728    subroutine check_surface_state(self _POSTARG_HORIZONTAL_IN_, repair, valid)
1729       class (type_fabm_model), intent(inout) :: self
1730       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1731       logical,                 intent(in)    :: repair
1732       logical,                 intent(out)   :: valid
1733
1734 #ifndef NDEBUG
1735       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'check_surface_stat
     e')
1736 #endif
1737
1738       call internal_check_horizontal_state(self, self%check_surface_state_job _POSTARG_HORIZONTAL_IN_, self%check_sur
     face_state_data, 1, self%surface_state_variables, repair, valid)
1739    end subroutine check_surface_state
1740
1741    subroutine internal_check_horizontal_state(self, job _POSTARG_HORIZONTAL_IN_, check_state_data, flag, state_variab
     les, repair, valid)
1742       class (type_fabm_model),                    intent(inout) :: self
1743       type (type_job),                            intent(in)    :: job
1744       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1745       type (type_check_state_data),               intent(in)    :: check_state_data(:)
1746       integer,                                    intent(in)    :: flag
1747       type (type_fabm_horizontal_state_variable), intent(inout) :: state_variables(:)
1748       logical,                                    intent(in)    :: repair
1749       logical,                                    intent(out)   :: valid
1750
1751       logical            :: valid_ranges
1752       integer            :: ivar, read_index
1753       real(rki)          :: value, minimum, maximum
1754       character(len=256) :: err
1755       _DECLARE_HORIZONTAL_INDICES_
1756 #ifdef _FABM_DEPTH_DIMENSION_INDEX_
1757       integer :: _VERTICAL_ITERATOR_
```

```
#endif

      self%cache_hz%repair = repair
      self%cache_hz%valid = .true.
      self%cache_hz%set_horizontal = .false.
      self%cache_hz%set_interior = .false.

      call process_horizontal_slice(job%first_task, self%domain, self%catalog, self%cache_fill_values, self%store, self%cache_hz _POSTARG_HORIZONTAL_IN_)

      valid = self%cache_hz%valid
      if (.not. (valid .or. repair)) return

      ! Quick bounds check for the common case where all values are valid.
      valid_ranges = .true.
      do ivar = 1, size(check_state_data)
         read_index = check_state_data(ivar)%index
         minimum = check_state_data(ivar)%minimum
         maximum = check_state_data(ivar)%maximum
         _HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
            value = self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index)
            if (value < minimum .or. value > maximum) valid_ranges = .false.
         _HORIZONTAL_LOOP_END_
      end do
      valid = valid .and. valid_ranges

      if (.not. valid_ranges) then
         ! One or more variables have out-of-bounds value(s)
         ! Either repair these by clipping or stop with an error message.
         do ivar = 1, size(check_state_data)
            read_index = check_state_data(ivar)%index
            minimum = check_state_data(ivar)%minimum
            maximum = check_state_data(ivar)%maximum

            _HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
               value = self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index)
               if (value < minimum) then
                  ! State variable value lies below prescribed minimum.
                  valid = .false.
                  if (.not. repair) then
                     write (unit=err,fmt='(a,e12.4,a,a,a,e12.4)') 'Value ',value,' of variable ', &
                                                                  & trim(state_variables(ivar)%name), &
                                                                  & ' below minimum value ',minimum
                     call log_message(err)
                     return
                  end if
                  self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index) = minimum
               elseif (value > maximum) then
                  ! State variable value exceeds prescribed maximum.
                  valid = .false.
                  if (.not. repair) then
                     write (unit=err,fmt='(a,e12.4,a,a,a,e12.4)') 'Value ',value,' of variable ', &
                                                                  & trim(state_variables(ivar)%name), &
                                                                  & ' above maximum value ',maximum
                     call log_message(err)
                     return
                  end if
                  self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(read_index) = maximum
               end if
            _HORIZONTAL_LOOP_END_
         end do
      end if

      if (self%cache_hz%set_horizontal .or. .not. valid_ranges) then
         do ivar = 1, size(state_variables)
            read_index = check_state_data(ivar)%index
            if (self%catalog%horizontal_sources(read_index) == data_source_fabm) then
               _HORIZONTAL_UNPACK_TO_GLOBAL_(self%cache_hz%read_hz, read_index, self%catalog%horizontal(state_variables(ivar)%target%catalog_index)%p, self%cache_hz, state_variables(ivar)%missing_value)
            end if
         end do
      end if

      if (self%cache_hz%set_interior) then
         ! One or more models have provided new values for an interior state variable [at the interface]

#ifdef _FABM_DEPTH_DIMENSION_INDEX_
         if (flag == 1) then
#  ifdef _FABM_VERTICAL_BOTTOM_TO_SURFACE_
            _VERTICAL_ITERATOR_ = self%domain%stop(_FABM_DEPTH_DIMENSION_INDEX_)
#  else
            _VERTICAL_ITERATOR_ = self%domain%start(_FABM_DEPTH_DIMENSION_INDEX_)
#  endif
         else
#  if _FABM_BOTTOM_INDEX_==0
#    ifdef _FABM_VERTICAL_BOTTOM_TO_SURFACE_
            _VERTICAL_ITERATOR_ = self%domain%start(_FABM_DEPTH_DIMENSION_INDEX_)
#    else
            _VERTICAL_ITERATOR_ = self%domain%stop(_FABM_DEPTH_DIMENSION_INDEX_)
#    endif
#  elif !defined(_HORIZONTAL_IS_VECTORIZED_)
            _VERTICAL_ITERATOR_ = self%domain%bottom_indices _INDEX_HORIZONTAL_LOCATION_
#  endif
         end if
#endif

         do ivar = 1, size(self%interior_state_variables)
            read_index = self%interior_state_variables(ivar)%target%read_indices%value
```

```
1854            if (self%catalog%interior_sources(read_index) == data_source_fabm) then
1855#if _FABM_BOTTOM_INDEX_==-1&&defined(_HORIZONTAL_IS_VECTORIZED_)
1856               if (flag == 1) then
1857#endif
1858
1859#ifdef _HORIZONTAL_IS_VECTORIZED_
1860#  ifdef _HAS_MASK_
1861                  self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_GLOBAL_INT
     ERIOR_(self%cache_hz%ipack(1:self%cache_hz%n)) = self%cache_hz%read(1:self%cache_hz%n, read_index)
1862#  else
1863                  _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
1864                     self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_GLOBAL_
     INTERIOR_(_START_+_I_-1) = self%cache_hz%read _INDEX_SLICE_PLUS_1_(read_index)
1865                  _HORIZONTAL_LOOP_END_
1866#  endif
1867#elif defined(_INTERIOR_IS_VECTORIZED_)
1868                  self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_LOCATION_
     = self%cache_hz%read(1,read_index)
1869#else
1870                  self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_LOCATION_
     = self%cache_hz%read(read_index)
1871#endif
1872
1873#if _FABM_BOTTOM_INDEX_==-1&&defined(_HORIZONTAL_IS_VECTORIZED_)
1874               else
1875                  ! Special case for bottom if vertical index of bottom point is variable.
1876                  _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
1877#  ifdef _HAS_MASK_
1878                     _VERTICAL_ITERATOR_ = self%domain%bottom_indices _INDEX_GLOBAL_HORIZONTAL_(self%cache_hz%ipack(_
     J_))
1879                     self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_GLOBAL_
     INTERIOR_(self%cache_hz%ipack(_J_)) = self%cache_hz%read _INDEX_SLICE_PLUS_1_(read_index)
1880#  else
1881                     _VERTICAL_ITERATOR_ = self%domain%bottom_indices _INDEX_GLOBAL_HORIZONTAL_(_START_+_J_-1)
1882                     self%catalog%interior(self%interior_state_variables(ivar)%target%catalog_index)%p _INDEX_GLOBAL_
     INTERIOR_(_START_+_I_-1) = self%cache_hz%read _INDEX_SLICE_PLUS_1_(read_index)
1883#  endif
1884                  _HORIZONTAL_LOOP_END_
1885               end if
1886#endif
1887            end if
1888         end do
1889      end if
1890   end subroutine internal_check_horizontal_state
1891
1892   subroutine get_surface_sources(self _POSTARG_HORIZONTAL_IN_, flux_pel, flux_sf)
1893      class (type_fabm_model),                            intent(inout)         :: self
1894      _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1895      real(rke) _DIMENSION_EXT_HORIZONTAL_SLICE_PLUS_1_, intent(out)           :: flux_pel
1896      real(rke) _DIMENSION_EXT_HORIZONTAL_SLICE_PLUS_1_, intent(out), optional :: flux_sf
1897
1898      integer :: i, k
1899      _DECLARE_HORIZONTAL_INDICES_
1900
1901#ifndef NDEBUG
1902      call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'get_surface_source
     s')
1903#  ifdef _HORIZONTAL_IS_VECTORIZED_
1904      call check_extents_2d(flux_pel, _STOP_ - _START_ + 1, size(self%interior_state_variables), 'get_surface_sources
     ', 'flux_pel', 'stop-start+1, # interior state variables')
1905      if (present(flux_sf)) call check_extents_2d(flux_sf, _STOP_ - _START_ + 1, size(self%surface_state_variables),
      'get_surface_sources', 'flux_sf', 'stop-start+1, # surface state variables')
1906#  else
1907      call check_extents_1d(flux_pel, size(self%interior_state_variables), 'get_surface_sources', 'flux_pel', '# inte
     rior state variables')
1908      if (present(flux_sf)) call check_extents_1d(flux_sf, size(self%surface_state_variables), 'get_surface_sources',
      'flux_sf', '# surface state variables')
1909#  endif
1910#endif
1911
1912      call process_horizontal_slice(self%get_surface_sources_job%first_task, self%domain, self%catalog, self%cache_fi
     ll_values, self%store, self%cache_hz _POSTARG_HORIZONTAL_IN_)
1913
1914      ! Compose surface fluxes for each interior state variable, combining model-specific contributions.
1915      flux_pel = 0.0_rke
1916      do i = 1, size(self%get_surface_sources_job%arg1_sources)
1917         k = self%get_surface_sources_job%arg1_sources(i)
1918         _HORIZONTAL_UNPACK_AND_ADD_TO_PLUS_1_(self%cache_hz%write_hz, k, flux_pel, i, self%cache_hz)
1919      end do
1920
1921      ! Compose total sources-sinks for each surface-bound state variable, combining model-specific contributions.
1922      if (present(flux_sf)) then
1923         flux_sf = 0.0_rke
1924         do i = 1, size(self%get_surface_sources_job%arg2_sources)
1925            k = self%get_surface_sources_job%arg2_sources(i)
1926            _HORIZONTAL_UNPACK_AND_ADD_TO_PLUS_1_(self%cache_hz%write_hz, k, flux_sf, i, self%cache_hz)
1927         end do
1928      end if
1929   end subroutine get_surface_sources
1930
1931   subroutine get_bottom_sources_rhs(self _POSTARG_HORIZONTAL_IN_, flux_pel, flux_ben)
1932      class (type_fabm_model),                            intent(inout) :: self
1933      _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1934      real(rke) _DIMENSION_EXT_HORIZONTAL_SLICE_PLUS_1_, intent(inout) :: flux_pel, flux_ben
1935
1936      integer :: i, k
1937      _DECLARE_HORIZONTAL_INDICES_
1938
```

```
1939 #ifndef NDEBUG
1940       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'get_bottom_sources
     _rhs')
1941 #  ifdef _HORIZONTAL_IS_VECTORIZED_
1942       call check_extents_2d(flux_pel, _STOP_ - _START_ + 1, size(self%interior_state_variables), 'get_bottom_sources_
     rhs', 'flux_pel', 'stop-start+1, # interior state variables')
1943       call check_extents_2d(flux_ben, _STOP_ - _START_ + 1, size(self%bottom_state_variables), 'get_bottom_sources_rh
     s', 'flux_ben', 'stop-start+1, # bottom state variables')
1944 #  else
1945       call check_extents_1d(flux_pel, size(self%interior_state_variables), 'get_bottom_sources_rhs', 'flux_pel', '# i
     nterior state variables')
1946       call check_extents_1d(flux_ben, size(self%bottom_state_variables), 'get_bottom_sources_rhs', 'flux_ben', '# bot
     tom state variables')
1947 #  endif
1948 #endif
1949
1950       call process_horizontal_slice(self%get_bottom_sources_job%first_task, self%domain, self%catalog, self%cache_fil
     l_values, self%store, self%cache_hz _POSTARG_HORIZONTAL_IN_)
1951
1952       ! Compose bottom fluxes for each interior state variable, combining model-specific contributions.
1953       do i = 1, size(self%get_bottom_sources_job%arg1_sources)
1954          k = self%get_bottom_sources_job%arg1_sources(i)
1955          _HORIZONTAL_UNPACK_AND_ADD_TO_PLUS_1_(self%cache_hz%write_hz, k, flux_pel, i, self%cache_hz)
1956       end do
1957
1958       ! Compose total sources-sinks for each bottom-bound state variable, combining model-specific contributions.
1959       do i = 1, size(self%get_bottom_sources_job%arg2_sources)
1960          k = self%get_bottom_sources_job%arg2_sources(i)
1961          _HORIZONTAL_UNPACK_AND_ADD_TO_PLUS_1_(self%cache_hz%write_hz, k, flux_ben, i, self%cache_hz)
1962       end do
1963    end subroutine get_bottom_sources_rhs
1964
1965    subroutine get_bottom_sources_ppdd(self _POSTARG_HORIZONTAL_IN_, pp, dd, benthos_offset)
1966       class (type_fabm_model),                            intent(inout) :: self
1967       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
1968       integer,                                            intent(in)    :: benthos_offset
1969       real(rke) _DIMENSION_EXT_HORIZONTAL_SLICE_PLUS_2_, intent(inout) :: pp, dd
1970
1971       integer                   :: icall, i, j, k, ncopy
1972       _DECLARE_HORIZONTAL_INDICES_
1973
1974 #ifndef NDEBUG
1975       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'get_bottom_sources
     _ppdd')
1976 #endif
1977
1978       call cache_pack(self%domain, self%catalog, self%cache_fill_values, self%get_bottom_sources_job%first_task, self
     %cache_hz _POSTARG_HORIZONTAL_IN_)
1979
1980       ncopy = 0
1981       do icall = 1, size(self%get_bottom_sources_job%first_task%calls)
1982          if (self%get_bottom_sources_job%first_task%calls(icall)%source == source_do_bottom) call self%get_bottom_sou
     rces_job%first_task%calls(icall)%model%do_bottom_ppdd(self%cache_hz, pp, dd, benthos_offset)
1983
1984          ! Copy outputs of interest to read cache so consecutive models can use it.
1985          _DO_CONCURRENT_(i,1 + ncopy,self%get_bottom_sources_job%first_task%calls(icall)%ncopy_hz + ncopy)
1986             j = self%get_bottom_sources_job%first_task%copy_commands_hz(i)%read_index
1987             k = self%get_bottom_sources_job%first_task%copy_commands_hz(i)%write_index
1988             _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(self%cache_hz)
1989                self%cache_hz%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(j) = self%cache_hz%write_hz _INDEX_HORIZONTAL_SL
     ICE_PLUS_1_(k)
1990             _HORIZONTAL_LOOP_END_
1991          end do
1992          ncopy = ncopy + self%get_bottom_sources_job%first_task%calls(icall)%ncopy_hz
1993       end do
1994
1995       call cache_unpack(self%get_bottom_sources_job%first_task, self%cache_hz, self%store _POSTARG_HORIZONTAL_IN_)
1996    end subroutine get_bottom_sources_ppdd
1997
1998    subroutine get_vertical_movement(self _POSTARG_INTERIOR_IN_, velocity)
1999       class (type_fabm_model),                intent(inout) :: self
2000       _DECLARE_ARGUMENTS_INTERIOR_IN_
2001       real(rke) _DIMENSION_EXT_SLICE_PLUS_1_, intent(out)   :: velocity
2002
2003       integer :: i, k
2004       _DECLARE_INTERIOR_INDICES_
2005
2006 #ifndef NDEBUG
2007       call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'get_vertical_movement'
     )
2008 #  ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
2009       call check_extents_2d(velocity, _STOP_ - _START_ + 1, size(self%interior_state_variables), 'get_vertical_moveme
     nt', 'velocity', 'stop-start+1, # interior state variables')
2010 #  else
2011       call check_extents_1d(velocity, size(self%interior_state_variables), 'get_vertical_movement', 'velocity', '# in
     terior state variables')
2012 #  endif
2013 #endif
2014
2015       call process_interior_slice(self%get_vertical_movement_job%first_task, self%domain, self%catalog, self%cache_fi
     ll_values, self%store, self%cache_int _POSTARG_INTERIOR_IN_)
2016
2017       ! Copy vertical velocities from write cache to output array provided by host
2018       do i = 1, size(self%get_vertical_movement_job%arg1_sources)
2019          k = self%get_vertical_movement_job%arg1_sources(i)
2020          _UNPACK_TO_PLUS_1_(self%cache_int%write, k, velocity, i, self%cache_int, 0.0_rke)
2021       end do
2022    end subroutine get_vertical_movement
```

```
2023
2024    subroutine get_interior_conserved_quantities(self _POSTARG_INTERIOR_IN_, sums)
2025       class (type_fabm_model),                intent(inout) :: self
2026       _DECLARE_ARGUMENTS_INTERIOR_IN_
2027       real(rke) _DIMENSION_EXT_SLICE_PLUS_1_, intent(out)   :: sums
2028
2029       integer :: i
2030       _DECLARE_INTERIOR_INDICES_
2031
2032 #ifndef NDEBUG
2033       call check_interior_location(self%domain%start, self%domain%stop _POSTARG_INTERIOR_IN_, 'get_interior_conserved
     _quantities')
2034 #  ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
2035       call check_extents_2d(sums, _STOP_ - _START_ + 1, size(self%conserved_quantities), 'get_interior_conserved_quan
     tities', 'sums', 'stop-start+1, # conserved quantities')
2036 #  else
2037       call check_extents_1d(sums, size(self%conserved_quantities), 'get_interior_conserved_quantities', 'sums', '# co
     nserved quantities')
2038 #  endif
2039 #endif
2040
2041       call process_interior_slice(self%get_interior_conserved_quantities_job%first_task, self%domain, self%catalog, s
     elf%cache_fill_values, self%store, self%cache_int _POSTARG_INTERIOR_IN_)
2042
2043       do i = 1, size(self%conserved_quantities)
2044          _UNPACK_TO_PLUS_1_(self%cache_int%write, self%conserved_quantities(i)%index, sums, i, self%cache_int, 0.0_rk
     e)
2045       end do
2046    end subroutine get_interior_conserved_quantities
2047
2048    subroutine get_horizontal_conserved_quantities(self _POSTARG_HORIZONTAL_IN_, sums)
2049       class (type_fabm_model),                              intent(inout) :: self
2050       _DECLARE_ARGUMENTS_HORIZONTAL_IN_
2051       real(rke) _DIMENSION_EXT_HORIZONTAL_SLICE_PLUS_1_, intent(out)   :: sums
2052
2053       integer :: i
2054       _DECLARE_HORIZONTAL_INDICES_
2055
2056 #ifndef NDEBUG
2057       call check_horizontal_location(self%domain%start, self%domain%stop _POSTARG_HORIZONTAL_IN_, 'get_horizontal_con
     served_quantities')
2058 #  ifdef _HORIZONTAL_IS_VECTORIZED_
2059       call check_extents_2d(sums, _STOP_ - _START_ + 1, size(self%conserved_quantities), 'get_horizontal_conserved_qu
     antities', 'sums', 'stop-start+1, # conserved quantities')
2060 #  else
2061       call check_extents_1d(sums, size(self%conserved_quantities), 'get_horizontal_conserved_quantities', 'sums', '#
     conserved quantities')
2062 #  endif
2063 #endif
2064
2065       call process_horizontal_slice(self%get_horizontal_conserved_quantities_job%first_task, self%domain, self%catalo
     g, self%cache_fill_values, self%store, self%cache_hz _POSTARG_HORIZONTAL_IN_)
2066
2067       do i = 1, size(self%conserved_quantities)
2068          _HORIZONTAL_UNPACK_TO_PLUS_1_(self%cache_hz%write_hz, self%conserved_quantities(i)%horizontal_index, sums, i
     , self%cache_hz, 0.0_rke)
2069       end do
2070    end subroutine get_horizontal_conserved_quantities
2071
2072    subroutine process_job(self, job _POSTARG_HORIZONTAL_LOCATION_RANGE_)
2073       class (type_fabm_model), intent(inout), target :: self
2074       type (type_job),         intent(in)            :: job
2075       _DECLARE_ARGUMENTS_HORIZONTAL_LOCATION_RANGE_
2076
2077       integer                   :: i
2078       type (type_task), pointer :: task
2079       _DECLARE_LOCATION_
2080
2081 #ifdef _FABM_DEPTH_DIMENSION_INDEX_
2082       ! Jobs must be applied across the entire depth range (if any),
2083       ! so we set vertical start and stop indices here.
2084       integer :: _VERTICAL_START_, _VERTICAL_STOP_
2085       _VERTICAL_START_ = self%domain%start(_FABM_DEPTH_DIMENSION_INDEX_)
2086       _VERTICAL_STOP_ = self%domain%stop(_FABM_DEPTH_DIMENSION_INDEX_)
2087 #endif
2088
2089       do i = 1, size(job%interior_store_prefill)
2090          if (job%interior_store_prefill(i)) then
2091             _BEGIN_OUTER_INTERIOR_LOOP_
2092                self%store%interior _INDEX_GLOBAL_INTERIOR_PLUS_1_(_START_:_STOP_, i) = self%store%interior_fill_value
     (i)
2093             _END_OUTER_INTERIOR_LOOP_
2094          end if
2095       end do
2096       do i = 1, size(job%horizontal_store_prefill)
2097          if (job%horizontal_store_prefill(i)) then
2098             _BEGIN_OUTER_HORIZONTAL_LOOP_
2099                self%store%horizontal _INDEX_GLOBAL_HORIZONTAL_PLUS_1_(_START_:_STOP_, i) = self%store%horizontal_fill
     _value(i)
2100             _END_OUTER_HORIZONTAL_LOOP_
2101          end if
2102       end do
2103
2104       task => job%first_task
2105       do while (associated(task))
2106          select case (task%operation)
2107          case (source_do)
2108             _BEGIN_OUTER_INTERIOR_LOOP_
```

```
2109                call process_interior_slice(task, self%domain, self%catalog, self%cache_fill_values, self%store, self%
     cache_int _POSTARG_INTERIOR_IN_)
2110             _END_OUTER_INTERIOR_LOOP_
2111          case (source_do_surface, source_do_bottom, source_do_horizontal)
2112             _BEGIN_OUTER_HORIZONTAL_LOOP_
2113                call process_horizontal_slice(task, self%domain, self%catalog, self%cache_fill_values, self%store, sel
     f%cache_hz _POSTARG_HORIZONTAL_IN_)
2114             _END_OUTER_HORIZONTAL_LOOP_
2115          case (source_do_column)
2116             _BEGIN_OUTER_VERTICAL_LOOP_
2117 #ifdef _FABM_DEPTH_DIMENSION_INDEX_
2118 #  if _FABM_BOTTOM_INDEX_==-1
2119 #    ifdef _FABM_VERTICAL_BOTTOM_TO_SURFACE_
2120                _VERTICAL_START_ = self%domain%bottom_indices _INDEX_HORIZONTAL_LOCATION_
2121 #    else
2122                _VERTICAL_STOP_ = self%domain%bottom_indices _INDEX_HORIZONTAL_LOCATION_
2123 #    endif
2124 #  endif
2125 #endif
2126                if (_IS_UNMASKED_(self%domain%mask_hz _INDEX_HORIZONTAL_LOCATION_)) call process_vertical_slice(task,
     self%domain, &
2127                     self%catalog, self%cache_fill_values, self%store, self%cache_vert _POSTARG_VERTICAL_IN_)
2128             _END_OUTER_VERTICAL_LOOP_
2129 #ifdef _FABM_DEPTH_DIMENSION_INDEX_
2130             _VERTICAL_START_ = self%domain%start(_FABM_DEPTH_DIMENSION_INDEX_)
2131             _VERTICAL_STOP_ = self%domain%stop(_FABM_DEPTH_DIMENSION_INDEX_)
2132 #endif
2133          end select
2134          task => task%next
2135       end do
2136    end subroutine process_job
2137
2138 #if _FABM_DIMENSION_COUNT_>1 || (_FABM_DIMENSION_COUNT_==1 && !defined(_FABM_DEPTH_DIMENSION_INDEX_))
2139    subroutine process_job_everywhere(self, job)
2140       class (type_fabm_model), intent(inout), target :: self
2141       type (type_job),         intent(in)            :: job
2142       integer :: _LOCATION_RANGE_
2143       istart__ = self%domain%start(1)
2144       istop__ = self%domain%stop(1)
2145 #  if _FABM_DIMENSION_COUNT_ > 1
2146       jstart__ = self%domain%start(2)
2147       jstop__ = self%domain%stop(2)
2148 #  endif
2149 #  if _FABM_DIMENSION_COUNT_ > 2
2150       kstart__ = self%domain%start(3)
2151       kstop__ = self%domain%stop(3)
2152 #  endif
2153       call process_job(self, job _POSTARG_HORIZONTAL_LOCATION_RANGE_)
2154    end subroutine process_job_everywhere
2155 #endif
2156
2157    subroutine prepare_inputs1(self, t)
2158       class (type_fabm_model),  intent(inout) :: self
2159       real(rke), optional,      intent(in)    :: t
2160
2161       class (type_expression), pointer :: expression
2162       _DECLARE_LOCATION_
2163
2164 #  if _FABM_DIMENSION_COUNT_ > 0
2165       integer :: _LOCATION_RANGE_
2166       istart__ = self%domain%start(1)
2167       istop__ = self%domain%stop(1)
2168 #  endif
2169 #  if _FABM_DIMENSION_COUNT_ > 1
2170       jstart__ = self%domain%start(2)
2171       jstop__ = self%domain%stop(2)
2172 #  endif
2173 #  if _FABM_DIMENSION_COUNT_ > 2
2174       kstart__ = self%domain%start(3)
2175       kstop__ = self%domain%stop(3)
2176 #  endif
2177
2178       call self%process(self%prepare_inputs_job)
2179
2180       if (present(t)) then
2181          ! The host has provided information about time. Use this to update moving averages, maxima (if any)
2182          expression => self%root%first_expression
2183          do while (associated(expression))
2184             select type (expression)
2185             class is (type_interior_temporal_mean)
2186                _ASSERT_(associated(self%catalog%interior(expression%in)%p), 'prepare_inputs1', 'source pointer of ' /
     / trim(expression%output_name) // ' not associated.')
2187                call expression%update(t, self%catalog%interior(expression%in)%p _POSTARG_LOCATION_RANGE_)
2188             class is (type_horizontal_temporal_mean)
2189                call update_horizontal_temporal_mean(expression)
2190             class is (type_horizontal_temporal_maximum)
2191                _ASSERT_(associated(self%catalog%horizontal(expression%in)%p), 'prepare_inputs1', 'source pointer of '
      // trim(expression%output_name) // ' not associated.')
2192                call expression%update(t, self%catalog%horizontal(expression%in)%p _POSTARG_HORIZONTAL_LOCATION_RANGE_
     )
2193             end select
2194             expression => expression%next
2195          end do
2196       end if
2197
2198    contains
2199
2200       subroutine update_horizontal_temporal_mean(expression)
```

```
2201          class (type_horizontal_temporal_mean), intent(inout) :: expression
2202
2203          integer  :: i
2204          real(rke) :: weight_right, frac_outside
2205
2206          if (expression%ioldest == -1) then
2207             ! Start of simulation; set entire history equal to current value.
2208             do i = 1, expression%n + 3
2209                _BEGIN_OUTER_VERTICAL_LOOP_
2210                   expression%history(_PREARG_HORIZONTAL_LOCATION_ i) = self%catalog%horizontal(expression%in)%p _INDEX_HORIZONTAL_LOCATION_
2211                _END_OUTER_VERTICAL_LOOP_
2212             end do
2213             expression%next_save_time = t + expression%period / expression%n
2214             expression%ioldest = 1
2215          end if
2216          do while (t >= expression%next_save_time)
2217             ! Weight for linear interpolation between last stored point and current point, to get at values for desired time.
2218             weight_right = (expression%next_save_time - expression%last_time) / (t - expression%last_time)
2219
2220             ! Remove contribution of oldest point from historical mean
2221             _BEGIN_OUTER_VERTICAL_LOOP_
2222                expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 2) = expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 2) &
2223                     - expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%ioldest) / expression%n
2224             _END_OUTER_VERTICAL_LOOP_
2225
2226             ! Linearly interpolate to desired time
2227             _BEGIN_OUTER_VERTICAL_LOOP_
2228                expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%ioldest) = (1.0_rke - weight_right)*expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 1) &
2229                     + weight_right*self%catalog%horizontal(expression%in)%p _INDEX_HORIZONTAL_LOCATION_
2230             _END_OUTER_VERTICAL_LOOP_
2231
2232             ! Add contribution of new point to historical mean
2233             _BEGIN_OUTER_VERTICAL_LOOP_
2234                expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 2) = expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 2) &
2235                     + expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%ioldest) / expression%n
2236             _END_OUTER_VERTICAL_LOOP_
2237
2238             ! Compute next time for which we want to store output
2239             expression%next_save_time = expression%next_save_time + expression%period / expression%n
2240
2241             ! Increment index for oldest time point
2242             expression%ioldest = expression%ioldest + 1
2243             if (expression%ioldest > expression%n) expression%ioldest = 1
2244          end do
2245
2246          ! Compute extent of time period outside history
2247          frac_outside = (t - (expression%next_save_time - expression%period / expression%n)) / expression%period
2248
2249          ! Store current value to enable linear interpolation to next output time in subsequent call.
2250          _BEGIN_OUTER_VERTICAL_LOOP_
2251             expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 1) = self%catalog%horizontal(expression%in)%p _INDEX_HORIZONTAL_LOCATION_
2252          _END_OUTER_VERTICAL_LOOP_
2253
2254          ! Set corrected running mean (move window by removing part of the start, and appending to the end)
2255          _BEGIN_OUTER_VERTICAL_LOOP_
2256             expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 3) = expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 2) &
2257                  + frac_outside * (-expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%ioldest) + expression%history(_PREARG_HORIZONTAL_LOCATION_ expression%n + 1))
2258          _END_OUTER_VERTICAL_LOOP_
2259
2260          expression%last_time = t
2261       end subroutine
2262    end subroutine prepare_inputs1
2263
2264    subroutine prepare_inputs2(self, t, year, month, day, seconds)
2265       class (type_fabm_model), intent(inout) :: self
2266       real(rke),               intent(in)    :: t
2267       integer,                 intent(in)    :: year, month, day
2268       real(rke),               intent(in)    :: seconds
2269
2270       call self%schedules%update(year, month, day, seconds)
2271       call prepare_inputs1(self, t)
2272    end subroutine prepare_inputs2
2273
2274    subroutine finalize_outputs(self)
2275       class (type_fabm_model),  intent(inout) :: self
2276
2277       call self%process(self%finalize_outputs_job)
2278    end subroutine finalize_outputs
2279
2280    subroutine classify_variables(self)
2281       class (type_fabm_model), intent(inout), target :: self
2282
2283       type (type_link),                                pointer :: link, newlink
2284       type (type_fabm_interior_state_variable),        pointer :: statevar
2285       type (type_fabm_horizontal_state_variable),      pointer :: hz_statevar
2286       type (type_fabm_interior_diagnostic_variable),   pointer :: diagvar
2287       type (type_fabm_horizontal_diagnostic_variable), pointer :: hz_diagvar
2288       type (type_fabm_conserved_quantity),             pointer :: consvar
2289       type (type_internal_variable),                   pointer :: object
2290       integer                                                  :: nstate, nstate_bot, nstate_surf, ndiag, ndiag_hz, n
```

```
cons
2291
2292        type (type_aggregate_variable_list)         :: aggregate_variable_list
2293        type (type_aggregate_variable),     pointer :: aggregate_variable
2294        type (type_set)                             :: dependencies, dependencies_hz, dependencies_scalar
2295        type (type_standard_variable_set)           :: standard_variable_set
2296        type (type_standard_variable_node), pointer :: standard_variable_node
2297
2298        ! Build a list of all master variables (those that not have been coupled)
2299        link => self%root%links%first
2300        do while (associated(link))
2301           if (associated(link%target, link%original)) &
2302              newlink => self%links_postcoupling%append(link%target, link%target%name)
2303           link => link%next
2304        end do
2305
2306        ! Get list of conserved quantities (map to universal=domain-independent variables where possible)
2307        aggregate_variable_list = collect_aggregate_variables(self%root)
2308        aggregate_variable => aggregate_variable_list%first
2309        do while (associated(aggregate_variable))
2310           if (associated(aggregate_variable%standard_variable%universal)) then
2311              if (aggregate_variable%standard_variable%universal%conserved) call standard_variable_set%add(aggregate_va
riable%standard_variable%universal)
2312           end if
2313           aggregate_variable => aggregate_variable%next
2314        end do
2315        call aggregate_variable_list%finalize()
2316
2317        ! Count number of conserved quantities and allocate an array for them.
2318        ncons = 0
2319        standard_variable_node => standard_variable_set%first
2320        do while (associated(standard_variable_node))
2321           ncons = ncons + 1
2322           standard_variable_node => standard_variable_node%next
2323        end do
2324        allocate(self%conserved_quantities(ncons))
2325
2326        ! Fill list of conserved quantities.
2327        ! This must be done before building the final authoratitive list of diagnostic variables,
2328        ! as the calls to append_data_pointer affect the global identifier of diagnostic variables
2329        ! by adding another pointer that must be set.
2330        ncons = 0
2331        standard_variable_node => standard_variable_set%first
2332        do while (associated(standard_variable_node))
2333           ncons = ncons + 1
2334           consvar => self%conserved_quantities(ncons)
2335           consvar%standard_variable => standard_variable_node%p
2336           consvar%name = trim(consvar%standard_variable%name)
2337           consvar%units = trim(consvar%standard_variable%units)
2338           consvar%long_name = trim(consvar%standard_variable%name)
2339           consvar%path = trim(consvar%standard_variable%name)
2340           select type (standard_variable => consvar%standard_variable)
2341           class is (type_universal_standard_variable)
2342              consvar%target => get_variable_by_standard_variable(self, standard_variable%in_interior())
2343              _ASSERT_(associated(consvar%target), 'classify_variables', 'Conserved quantity ' // trim(standard_variabl
e%name) // ' not found in interior.')
2344              consvar%target_hz => get_variable_by_standard_variable(self, standard_variable%at_interfaces())
2345              _ASSERT_(associated(consvar%target_hz), 'classify_variables', 'Conserved quantity ' // trim(standard_vari
able%name) // ' not found at interfaces.')
2346           end select
2347           consvar%missing_value = consvar%target%missing_value
2348           standard_variable_node => standard_variable_node%next
2349        end do
2350        call standard_variable_set%finalize()
2351
2352        ! From this point on, variables will stay as they are.
2353        ! Coupling is done, and the framework will not add further read indices.
2354
2355        ! Count number of interior variables in various categories.
2356        nstate = 0
2357        ndiag  = 0
2358        nstate_bot  = 0
2359        nstate_surf = 0
2360        ndiag_hz    = 0
2361        link => self%links_postcoupling%first
2362        do while (associated(link))
2363           object => link%target
2364           _ASSERT_(object%source /= source_state .or. object%write_indices%is_empty(), 'classify_variables', 'variable
 ' // trim(object%name) // ' has source_state and one or more write indices.')
2365           select case (object%domain)
2366           case (domain_interior)
2367              select case (object%source)
2368              case (source_unknown)   ! Interior dependency
2369              case (source_state)     ! Interior state variable
2370                 select case (object%presence)
2371                 case (presence_internal)
2372                    nstate = nstate + 1
2373                    call object%state_indices%set_value(nstate)
2374                 case (presence_external_required)
2375                    call fatal_error('classify_variables', &
2376                       'Variable ' // trim(link%name) // ' must be coupled to an existing state variable.')
2377                 case (presence_external_optional)
2378                    ! Optional interior state variable that was not coupled
2379                    ! Ðemote to simple optional dependency; any sources will be ignored.
2380                    object%source = source_unknown
2381                 end select
2382              case default            ! Interior diagnostic variable
2383                 ndiag = ndiag + 1
```

```
2384            end select
2385         case (domain_horizontal, domain_surface, domain_bottom)
2386            select case (object%source)
2387            case (source_unknown)   ! Horizontal dependency
2388            case (source_state)     ! Horizontal state variable
2389               select case (object%presence)
2390               case (presence_internal)
2391                  select case (object%domain)
2392                     case (domain_bottom)
2393                        nstate_bot = nstate_bot + 1
2394                        call object%state_indices%set_value(nstate_bot)
2395                     case (domain_surface)
2396                        nstate_surf = nstate_surf + 1
2397                        call object%state_indices%set_value(nstate_surf)
2398                  end select
2399               case (presence_external_required)
2400                  call fatal_error('classify_variables', &
2401                     'Variable ' // trim(link%name) // ' must be coupled to an existing state variable.')
2402               case (presence_external_optional)
2403                  ! Optional horizontal state variable that was not coupled
2404                  ! Demote to simple optional dependency; any sources will be ignored.
2405                  object%source = source_unknown
2406               end select
2407            case default            ! Horizontal diagnostic variable
2408               ndiag_hz = ndiag_hz + 1
2409            end select
2410         end select
2411         link => link%next
2412      end do
2413
2414      ! Allocate arrays with variable information that will be accessed by the host model.
2415      allocate(self%interior_state_variables       (nstate))
2416      allocate(self%bottom_state_variables         (nstate_bot))
2417      allocate(self%surface_state_variables        (nstate_surf))
2418      allocate(self%interior_diagnostic_variables  (ndiag))
2419      allocate(self%horizontal_diagnostic_variables(ndiag_hz))
2420
2421      allocate(self%get_interior_sources_job%arg1_sources(nstate))
2422      allocate(self%get_surface_sources_job%arg1_sources(nstate), self%get_surface_sources_job%arg2_sources(nstate_su
     rf))
2423      allocate(self%get_bottom_sources_job%arg1_sources(nstate), self%get_bottom_sources_job%arg2_sources(nstate_bot)
     )
2424      allocate(self%get_vertical_movement_job%arg1_sources(nstate))
2425
2426      ! Build lists of state variable and diagnostic variables.
2427      nstate = 0
2428      ndiag  = 0
2429      nstate_bot  = 0
2430      nstate_surf = 0
2431      ndiag_hz    = 0
2432      link => self%links_postcoupling%first
2433      do while (associated(link))
2434         object => link%target
2435         select case (link%target%domain)
2436         case (domain_interior)
2437            select case (object%source)
2438            case (source_unknown)   ! Interior dependency
2439            case (source_state)     ! Interior state variable
2440               if (object%presence == presence_internal) then
2441                  nstate = nstate + 1
2442                  statevar => self%interior_state_variables(nstate)
2443                  call copy_variable_metadata(object, statevar)
2444                  if (associated(object%standard_variables%first)) then
2445                     select type (standard_variable => object%standard_variables%first%p)
2446                     class is (type_interior_standard_variable)
2447                        statevar%standard_variable => standard_variable
2448                     end select
2449                  end if
2450                  statevar%initial_value             = object%initial_value
2451                  statevar%no_precipitation_dilution = object%no_precipitation_dilution
2452                  statevar%no_river_dilution         = object%no_river_dilution
2453                  call object%sms_sum%target%write_indices%append(self%get_interior_sources_job%arg1_sources(nstate))

2454                  call object%surface_flux_sum%target%write_indices%append(self%get_surface_sources_job%arg1_sources(
     nstate))
2455                  call object%bottom_flux_sum%target%write_indices%append(self%get_bottom_sources_job%arg1_sources(ns
     tate))
2456                  call object%movement_sum%target%write_indices%append(self%get_vertical_movement_job%arg1_sources(ns
     tate))
2457               end if
2458            case default            ! Interior diagnostic variable
2459               ndiag = ndiag + 1
2460               diagvar => self%interior_diagnostic_variables(ndiag)
2461               call copy_variable_metadata(object, diagvar)
2462               if (associated(object%standard_variables%first)) then
2463                  select type (standard_variable => object%standard_variables%first%p)
2464                  class is (type_interior_standard_variable)
2465                     diagvar%standard_variable => standard_variable
2466                  end select
2467               end if
2468               diagvar%save = diagvar%output /= output_none
2469               diagvar%source = object%source
2470            end select
2471         case (domain_horizontal, domain_surface, domain_bottom)
2472            select case (object%source)
2473            case (source_unknown)   ! Horizontal dependency
2474            case (source_state)     ! Horizontal state variable
2475               if (object%presence == presence_internal) then
```

```
2476                    select case (object%domain)
2477                    case (domain_bottom)
2478                       nstate_bot = nstate_bot + 1
2479                       hz_statevar => self%bottom_state_variables(nstate_bot)
2480                       call object%sms_sum%target%write_indices%append(self%get_bottom_sources_job%arg2_sources(nstate_
    bot))
2481                    case (domain_surface)
2482                       nstate_surf = nstate_surf + 1
2483                       hz_statevar => self%surface_state_variables(nstate_surf)
2484                       call object%sms_sum%target%write_indices%append(self%get_surface_sources_job%arg2_sources(nstate
    _surf))
2485                    case default
2486                       hz_statevar => null()
2487                    end select
2488                    call copy_variable_metadata(object, hz_statevar)
2489                    if (associated(object%standard_variables%first)) then
2490                       select type (standard_variable => object%standard_variables%first%p)
2491                       class is (type_horizontal_standard_variable)
2492                          hz_statevar%standard_variable => standard_variable
2493                       end select
2494                    end if
2495                    hz_statevar%initial_value = object%initial_value
2496                 end if
2497              case default            ! Horizontal diagnostic variable
2498                 ndiag_hz = ndiag_hz + 1
2499                 hz_diagvar => self%horizontal_diagnostic_variables(ndiag_hz)
2500                 call copy_variable_metadata(object, hz_diagvar)
2501                 if (associated(object%standard_variables%first)) then
2502                    select type (standard_variable => object%standard_variables%first%p)
2503                    class is (type_horizontal_standard_variable)
2504                       hz_diagvar%standard_variable => standard_variable
2505                    end select
2506                 end if
2507                 hz_diagvar%save = hz_diagvar%output /= output_none
2508                 hz_diagvar%source = object%source
2509              end select
2510           end select
2511           link => link%next
2512        end do
2513
2514        ! Create lists of variables that may be provided by the host model.
2515        ! These lists include external dependencies, as well as the model's own state variables,
2516        ! which may be overridden by the host.
2517        link => self%root%links%first
2518        do while (associated(link))
2519           object =>link%target
2520           if (.not. object%read_indices%is_empty() .and. &
2521               .not. (object%presence == presence_external_optional .and. object%source == source_state)) then
2522              select case (object%domain)
2523                 case (domain_interior);                                  call dependencies%add(link%name)
2524                 case (domain_horizontal, domain_surface, domain_bottom); call dependencies_hz%add(link%name)
2525                 case (domain_scalar);                                    call dependencies_scalar%add(link%name)
2526              end select
2527
2528              standard_variable_node => object%standard_variables%first
2529              do while (associated(standard_variable_node))
2530                 if (standard_variable_node%p%name /= '') then
2531                    select case (object%domain)
2532                       case (domain_interior);                                  call dependencies%add(standard_variable
    _node%p%name)
2533                       case (domain_horizontal, domain_surface, domain_bottom); call dependencies_hz%add(standard_varia
    ble_node%p%name)
2534                       case (domain_scalar);                                    call dependencies_scalar%add(standard_v
    ariable_node%p%name)
2535                    end select
2536                 end if
2537                 standard_variable_node => standard_variable_node%next
2538              end do
2539           end if
2540           link => link%next
2541        end do
2542        call dependencies%to_array(self%dependencies)
2543        call dependencies_hz%to_array(self%dependencies_hz)
2544        call dependencies_scalar%to_array(self%dependencies_scalar)
2545        call dependencies%finalize()
2546        call dependencies_hz%finalize()
2547        call dependencies_scalar%finalize()
2548     contains
2549        subroutine copy_variable_metadata(internal_variable, external_variable)
2550           class (type_fabm_variable),    intent(inout)      :: external_variable
2551           type (type_internal_variable), intent(in), target :: internal_variable
2552
2553           class (type_base_model), pointer :: owner
2554           external_variable%name            = get_safe_name(internal_variable%name)
2555           external_variable%long_name       = internal_variable%long_name
2556           external_variable%local_long_name = internal_variable%long_name
2557           external_variable%units           = internal_variable%units
2558           external_variable%path            = internal_variable%name
2559           external_variable%minimum         = internal_variable%minimum
2560           external_variable%maximum         = internal_variable%maximum
2561           external_variable%missing_value   = internal_variable%missing_value
2562           external_variable%output          = internal_variable%output
2563           external_variable%target          => internal_variable
2564
2565           ! Prepend long names of ancestor models to long name of variable.
2566           owner => internal_variable%owner
2567           do while (associated(owner%parent))
2568              external_variable%long_name = trim(owner%long_name) // ' ' // trim(external_variable%long_name)
```

```
2569             owner => owner%parent
2570          end do
2571
2572          call external_variable%properties%update(internal_variable%properties)
2573       end subroutine
2574    end subroutine classify_variables
2575
2576    subroutine require_flux_computation(self, link_list, domain)
2577       type (type_job),       intent(inout) :: self
2578       type (type_link_list), intent(in)    :: link_list
2579       integer,               intent(in)    :: domain
2580
2581       type (type_link), pointer :: link
2582
2583       link => link_list%first
2584       do while (associated(link))
2585          if (link%target%source == source_state) then
2586             ! This is a state variable
2587             select case (domain)
2588             case (domain_interior)
2589                if (link%target%domain == domain_interior .and. associated(link%target%sms_sum)) &
2590                   call self%request_variable(link%target%sms_sum%target)
2591             case (domain_bottom)
2592                if (link%target%domain == domain_bottom .and. associated(link%target%sms_sum)) then
2593                   call self%request_variable(link%target%sms_sum%target)
2594                elseif (link%target%domain == domain_interior .and. associated(link%target%bottom_flux_sum)) then
2595                   call self%request_variable(link%target%bottom_flux_sum%target)
2596                end if
2597             case (domain_surface)
2598                if (link%target%domain == domain_surface .and. associated(link%target%sms_sum)) then
2599                   call self%request_variable(link%target%sms_sum%target)
2600                elseif (link%target%domain == domain_interior .and. associated(link%target%surface_flux_sum)) then
2601                   call self%request_variable(link%target%surface_flux_sum%target)
2602                end if
2603             case (domain_interior + 999)
2604                if (link%target%domain == domain_interior .and. associated(link%target%movement_sum)) &
2605                   call self%request_variable(link%target%movement_sum%target)
2606             end select
2607          end if
2608          link => link%next
2609       end do
2610    end subroutine require_flux_computation
2611
2612    subroutine require_call_all_with_state(self, link_list, domain, source)
2613       type (type_job),       intent(inout) :: self
2614       type (type_link_list), intent(in)    :: link_list
2615       integer,               intent(in)    :: domain
2616       integer,               intent(in)    :: source
2617
2618       type (type_link), pointer :: link
2619
2620       link => link_list%first
2621       do while (associated(link))
2622          if (link%target%domain == domain .and. link%original%source == source_state .and. link%target%source == sour
     ce_state) &
2623             call self%request_call(link%original%owner, source)
2624          link => link%next
2625       end do
2626    end subroutine require_call_all_with_state
2627
2628    subroutine create_catalog(self)
2629       class (type_fabm_model), intent(inout) :: self
2630
2631       integer                   :: i
2632       type (type_link), pointer :: link
2633
2634       ! Add all state variables to the catalog and read cache in the order the host is likely to
2635       ! have them in memory. This hopefully speeds up access (cache hits).
2636       do i = 1, size(self%interior_state_variables)
2637          call self%variable_register%add_to_catalog(self%interior_state_variables(i)%target)
2638          call self%variable_register%add_to_read_cache(self%interior_state_variables(i)%target)
2639       end do
2640       do i = 1, size(self%bottom_state_variables)
2641          call self%variable_register%add_to_catalog(self%bottom_state_variables(i)%target)
2642          call self%variable_register%add_to_read_cache(self%bottom_state_variables(i)%target)
2643       end do
2644       do i = 1, size(self%surface_state_variables)
2645          call self%variable_register%add_to_catalog(self%surface_state_variables(i)%target)
2646          call self%variable_register%add_to_read_cache(self%surface_state_variables(i)%target)
2647       end do
2648
2649       ! Add all remaining variables to the catalog
2650       link => self%links_postcoupling%first
2651       do while (associated(link))
2652          call self%variable_register%add_to_catalog(link%target)
2653          link => link%next
2654       end do
2655
2656       allocate(self%catalog%interior(self%variable_register%catalog%interior%count))
2657       allocate(self%catalog%horizontal(self%variable_register%catalog%horizontal%count))
2658       allocate(self%catalog%scalar(self%variable_register%catalog%scalar%count))
2659
2660       ! Allocate and initialize arrays that store the source (host, fabm, user) of all data.
2661       allocate(self%catalog%interior_sources(size(self%catalog%interior)))
2662       allocate(self%catalog%horizontal_sources(size(self%catalog%horizontal)))
2663       allocate(self%catalog%scalar_sources(size(self%catalog%scalar)))
2664       self%catalog%interior_sources = data_source_none
2665       self%catalog%horizontal_sources = data_source_none
```

```
2666       self%catalog%scalar_sources = data_source_none
2667    end subroutine create_catalog
2668
2669    function get_cache_fill_values(variable_register) result(cache_fill_values)
2670       type (type_global_variable_register), intent(in) :: variable_register
2671       type (type_cache_fill_values)                    :: cache_fill_values
2672
2673       call collect(variable_register%read_cache%interior,    cache_fill_values%read,             use_missing=.false.)
2674       call collect(variable_register%read_cache%horizontal,  cache_fill_values%read_hz,          use_missing=.false.)
2675       call collect(variable_register%read_cache%scalar,      cache_fill_values%read_scalar,      use_missing=.false.)
2676       call collect(variable_register%write_cache%interior,   cache_fill_values%write,            use_missing=.false.)
2677       call collect(variable_register%write_cache%horizontal, cache_fill_values%write_hz,         use_missing=.false.)
2678       call collect(variable_register%write_cache%interior,   cache_fill_values%write_missing,    use_missing=.true.)
2679       call collect(variable_register%write_cache%horizontal, cache_fill_values%write_hz_missing, use_missing=.true.)
2680    contains
2681       subroutine collect(variable_list, values, use_missing)
2682          type (type_variable_list), intent(in)  :: variable_list
2683          real(rki), allocatable,    intent(out) :: values(:)
2684          logical,                   intent(in)  :: use_missing
2685
2686          integer                             :: i
2687          type (type_variable_node), pointer :: variable_node
2688
2689          allocate(values(variable_list%count))
2690          variable_node => variable_list%first
2691          do i = 1, size(values)
2692             if (use_missing) then
2693                values(i) = variable_node%target%missing_value
2694             else
2695                values(i) = variable_node%target%prefill_value
2696             end if
2697             variable_node => variable_node%next
2698          end do
2699       end subroutine
2700    end function
2701
2702    subroutine create_store(self)
2703       class (type_fabm_model), intent(inout), target :: self
2704
2705       type (type_variable_node),                pointer :: variable_node
2706       real(rke) _ATTRIBUTES_GLOBAL_,            pointer :: pdata
2707       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, pointer :: pdata_hz
2708
2709       ! Allocate memory for persistent store
2710 #if _FABM_DIMENSION_COUNT_==0
2711       call allocate_store()
2712 #elif _FABM_DIMENSION_COUNT_==1
2713       call allocate_store(self%domain%shape(1))
2714 #elif _FABM_DIMENSION_COUNT_==2
2715       call allocate_store(self%domain%shape(1), self%domain%shape(2))
2716 #else
2717       call allocate_store(self%domain%shape(1), self%domain%shape(2), self%domain%shape(3))
2718 #endif
2719
2720       ! Collect missing values in array for faster access. These will be used to fill masked parts of outputs.
2721       call collect_fill_values(self%variable_register%store%interior,   self%store%interior_fill_value,      use_miss
      ing=.false.)
2722       call collect_fill_values(self%variable_register%store%horizontal, self%store%horizontal_fill_value,    use_miss
      ing=.false.)
2723       call collect_fill_values(self%variable_register%store%interior,   self%store%interior_missing_value,   use_miss
      ing=.true.)
2724       call collect_fill_values(self%variable_register%store%horizontal, self%store%horizontal_missing_value, use_miss
      ing=.true.)
2725
2726       call reset_store(self)
2727
2728       ! Register data fields from persistent store in catalog.
2729       variable_node => self%variable_register%catalog%interior%first
2730       do while (associated(variable_node))
2731          if (variable_node%target%store_index > 0) then
2732             ! Note: we first assign to the pointer below to ensure ifort 15 recognizes its contiguity when _FABM_CONT
      IGUOUS is set
2733             pdata => self%store%interior(_PREARG_LOCATION_DIMENSIONS_ variable_node%target%store_index)
2734             call self%link_interior_data(variable_node%target, pdata, source=data_source_fabm)
2735          end if
2736          variable_node => variable_node%next
2737       end do
2738       variable_node => self%variable_register%catalog%horizontal%first
2739       do while (associated(variable_node))
2740          if (variable_node%target%store_index > 0) then
2741             ! Note: we first assign to the pointer below to ensure ifort 15 recognizes its contiguity when _FABM_CONT
      IGUOUS is set
2742             pdata_hz => self%store%horizontal(_PREARG_HORIZONTAL_LOCATION_DIMENSIONS_ variable_node%target%store_inde
      x)
2743             call self%link_horizontal_data(variable_node%target, pdata_hz, source=data_source_fabm)
2744          end if
2745          variable_node => variable_node%next
2746       end do
2747
2748    contains
2749
2750       subroutine allocate_store(_LOCATION_)
2751          _DECLARE_ARGUMENTS_LOCATION_
```

```
2752          allocate(self%store%interior(_PREARG_LOCATION_ 0:self%variable_register%store%interior%count))
2753          allocate(self%store%horizontal(_PREARG_HORIZONTAL_LOCATION_ 0:self%variable_register%store%horizontal%count)
      )
2754       end subroutine
2755
2756       subroutine collect_fill_values(variable_list, values, use_missing)
2757          type (type_variable_list), intent(in)  :: variable_list
2758          real(rke), allocatable,    intent(out) :: values(:)
2759          logical,                   intent(in)  :: use_missing
2760
2761          integer                             :: i
2762          type (type_variable_node), pointer :: variable_node
2763
2764          allocate(values(variable_list%count))
2765          variable_node => variable_list%first
2766          do i = 1, size(values)
2767             if (use_missing) then
2768                values(i) = variable_node%target%missing_value
2769             else
2770                values(i) = variable_node%target%prefill_value
2771             end if
2772             variable_node => variable_node%next
2773          end do
2774       end subroutine
2775
2776    end subroutine create_store
2777
2778    subroutine reset_store(self)
2779       class (type_fabm_model), intent(inout) :: self
2780
2781       integer   :: i
2782       real(rke) :: fill_value
2783
2784       ! Initialize persistent store entries to fill value.
2785       ! For constant outputs, their values will be set here, and never touched again.
2786       ! The intermediate variable fill_value is used to prevent ifort 19.2 from putting large temporary arrays on the
      stack
2787       do i = 1, self%variable_register%store%interior%count
2788          fill_value = self%store%interior_fill_value(i)
2789          self%store%interior(_PREARG_LOCATION_DIMENSIONS_ i) = fill_value
2790       end do
2791       do i = 1, self%variable_register%store%horizontal%count
2792          fill_value = self%store%horizontal_fill_value(i)
2793          self%store%horizontal(_PREARG_HORIZONTAL_LOCATION_DIMENSIONS_ i) = fill_value
2794       end do
2795    end subroutine
2796
2797    recursive subroutine merge_indices(model, log_unit)
2798       class (type_base_model), intent(inout)        :: model
2799       integer,                 intent(in), optional :: log_unit
2800
2801       type (type_model_list_node), pointer :: child
2802
2803       select type (model)
2804       class is (type_reduction_operator)
2805          call model%merge_components(log_unit)
2806       end select
2807
2808       ! Process children
2809       child => model%children%first
2810       do while (associated(child))
2811          call merge_indices(child%model, log_unit)
2812          child => child%next
2813       end do
2814    end subroutine merge_indices
2815
2816    subroutine filter_expressions(self)
2817       class (type_fabm_model),intent(inout) :: self
2818
2819       class (type_expression),              pointer :: current, previous, next
2820       class (type_depth_integral),          pointer :: integral
2821       class (type_bounded_depth_integral), pointer :: bounded_integral
2822       logical                                       :: filter
2823
2824       previous => null()
2825       current => self%root%first_expression
2826       do while (associated(current))
2827          filter = .false.
2828          select type (current)
2829          class is (type_vertical_integral)
2830             if (current%minimum_depth == 0._rki .and. current%maximum_depth == huge(current%maximum_depth)) then
2831                allocate(integral)
2832             else
2833                allocate(bounded_integral)
2834                bounded_integral%minimum_depth = current%minimum_depth
2835                bounded_integral%maximum_depth = current%maximum_depth
2836                integral => bounded_integral
2837             end if
2838             integral%average = current%average
2839             call self%root%add_child(integral, trim(current%output_name) // '_calculator')
2840             call integral%request_coupling(integral%id_input, current%input_name)
2841             call self%root%request_coupling(current%output_name, integral%id_output%link%target%name)
2842             filter = .true.
2843          end select
2844
2845          ! If FABM handles this expression internally, remove it from the list.
2846          next => current%next
2847          if (filter) then
```

```
            if (associated(previous)) then
               previous%next => next
            else
               self%root%first_expression => next
            end if
            deallocate(current)
         else
            previous => current
         end if
         current => next
      end do
   end subroutine filter_expressions

   function get_variable_by_name(self, name, domain) result(variable)
      class (type_fabm_model), intent(in)    :: self
      character(len=*),        intent(in)    :: name
      integer,                 intent(in)    :: domain
      type (type_internal_variable), pointer :: variable

      type (type_link), pointer :: link

      variable => null()

      link => self%root%links%first
      do while (associated(link))
         if (iand(link%target%domain, domain) /= 0) then
            if (link%name == name .or. get_safe_name(link%name) == name) then
               variable => link%target
               return
            end if
         end if
         link => link%next
      end do

      ! Name not found among variable names. Now try standard names that are in use.
      link => self%root%links%first
      do while (associated(link))
         if (iand(link%target%domain, domain) /= 0 .and. link%target%standard_variables%contains(name)) then
            variable => link%target
            return
         end if
         link => link%next
      end do
   end function get_variable_by_name

   function get_variable_by_standard_variable(self, standard_variable) result(variable)
      class (type_fabm_model), intent(in)         :: self
      class (type_base_standard_variable), target :: standard_variable
      type (type_internal_variable), pointer :: variable

      type (type_link), pointer :: link

      variable => null()
      link => self%root%links%first
      do while (associated(link))
         if (link%target%standard_variables%contains(standard_variable)) then
            variable => link%target
            return
         end if
         link => link%next
      end do
      if (standard_variable%aggregate_variable) then
         select type (standard_variable)
         class is (type_interior_standard_variable)
            variable => self%root%find_object('zero')
         class is (type_horizontal_standard_variable)
            variable => self%root%find_object('zero_hz')
         end select
      end if
   end function get_variable_by_standard_variable

end module fabm

!-----------------------------------------------------------------------
! Copyright Bolding & Bruggeman ApS (GNU Public License - www.gnu.org)
!-----------------------------------------------------------------------
```