```
#include "fabm_driver.h"

! ======================================================================
! fabm_types --- Derived types and procedures for use by biogeochemical models
! ----------------------------------------------------------------------
! This module contains the derived types and procedures that are used for
! communication between biogeochemical models and FABM. This module provides
! types for storing model data (e.g., metadata for variables and parameters),
! and logic for registration of model objects (state and diagnostic variables),
! retrieval of model settings (parameter values) and coupling.
! ======================================================================

module fabm_types

   use fabm_parameters, rk=>rki
   use fabm_standard_variables, type_bulk_standard_variable => type_universal_standard_variable, &
      type_universal_standard_variable => type_universal_standard_variable
   use fabm_properties
   use fabm_driver, only: driver

   implicit none

   private

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

   ! Base data type for biogeochemical models.
   public type_base_model

   ! Expose symbols defined in fabm_standard_variables module
   public standard_variables
   public type_interior_standard_variable, type_horizontal_standard_variable, type_global_standard_variable, &
      type_universal_standard_variable, type_bottom_standard_variable, type_surface_standard_variable, type_domain_specific_standard_variable, &
      type_standard_variable_node, type_base_standard_variable, type_standard_variable_set

   ! Variable identifier types used by biogeochemical models
   public type_variable_id
   public type_diagnostic_variable_id, type_horizontal_diagnostic_variable_id, &
      type_surface_diagnostic_variable_id, type_bottom_diagnostic_variable_id
   public type_state_variable_id, type_surface_state_variable_id, type_bottom_state_variable_id
   public type_dependency_id, type_surface_dependency_id, type_bottom_dependency_id, type_horizontal_dependency_id, &
      type_global_dependency_id
   public type_add_id, type_horizontal_add_id

   ! Data types and procedures for variable management - used by FABM internally only.
   public type_link, type_link_list, type_link_pointer, type_variable_node, type_variable_set, type_variable_list
   public type_internal_variable
   public type_cache, type_interior_cache, type_horizontal_cache, type_vertical_cache
   public type_model_list, type_model_list_node

   public get_free_unit
   public get_safe_name
   public source2string

   public type_expression, type_interior_expression, type_horizontal_expression

   public get_aggregate_variable_access, type_aggregate_variable_access, type_contribution

   public type_coupling_task

   ! For backward compatibility (20200302, pre 1.0)
   public type_bulk_standard_variable

   integer, parameter, public :: attribute_length = 256

   public rk, rke

   integer, parameter, public :: domain_interior   = 4, &
                                 domain_horizontal = 8, &
                                 domain_scalar     = 16, &
                                 domain_bottom     = 9, &
                                 domain_surface    = 10

   integer, parameter, public :: source_unknown                  =  0, &
                                 source_do                       =  1, &
                                 source_do_column                =  2, &
                                 source_do_horizontal            =  3, &
                                 source_do_bottom                =  4, &
                                 source_do_surface               =  5, &
                                 source_constant                 =  6, &
                                 source_none                     =  6, &
                                 source_get_vertical_movement    =  7, &
                                 source_initialize_state         =  8, &
                                 source_initialize_surface_state =  9, &
                                 source_initialize_bottom_state  = 10, &
                                 source_check_state              = 11, &
                                 source_check_surface_state      = 12, &
                                 source_check_bottom_state       = 13, &
                                 source_get_light_extinction     = 14, &
                                 source_get_drag                 = 15, &
                                 source_get_albedo               = 16, &
                                 source_external                 = 17, &
                                 source_state                    = 18

```

```
   integer, parameter, public :: presence_internal          = 1, &
                                 presence_external_required = 2, &
                                 presence_external_optional = 6

   integer, parameter, public :: prefill_none           = 0, &
                                 prefill_constant       = -1, &
                                 prefill_previous_value = -2

   integer, parameter, public :: access_none       = 0, &
                                 access_read       = 1, &
                                 access_set_source = 2, &
                                 access_state      = ior(access_read,access_set_source)

   integer, parameter, public :: store_index_none = -1

   integer, parameter, public :: operator_assign = 0, &
                                 operator_add    = 1, &
                                 operator_merge_forbidden = 256

   integer, parameter, public :: output_none                 = 0, &
                                 output_instantaneous        = 1, &
                                 output_time_integrated      = 2, &
                                 output_time_step_averaged   = 4, &
                                 output_time_step_integrated = 8

   ! -----------------------------------------------------------------------
   ! Data types for pointers to variable values
   ! -----------------------------------------------------------------------

   type type_integer_pointer
      integer, pointer :: p => null()
   end type

   type type_real_pointer
      real(rk), pointer :: p => null()
   end type

   type type_real_pointer_set
      type (type_real_pointer), allocatable :: pointers(:)
   contains
      procedure :: append    => real_pointer_set_append
      procedure :: extend    => real_pointer_set_extend
      procedure :: set_value => real_pointer_set_set_value
   end type

   type type_integer_pointer_set
      type (type_integer_pointer), allocatable :: pointers(:)
      integer                                  :: value = -1
   contains
      procedure :: append    => integer_pointer_set_append
      procedure :: extend    => integer_pointer_set_extend
      procedure :: set_value => integer_pointer_set_set_value
      procedure :: is_empty  => integer_pointer_set_is_empty
      procedure :: finalize  => integer_pointer_set_finalize
   end type

   ! -----------------------------------------------------------------------
   ! Data types for coupling tasks
   ! -----------------------------------------------------------------------

   type type_coupling_task
      type (type_link), pointer                    :: slave       => null()
      character(len=attribute_length)              :: master_name = ''
      class (type_domain_specific_standard_variable), pointer :: master_standard_variable => null()
      logical                                      :: user_specified = .false.
      class (type_coupling_task), pointer          :: previous    => null()
      class (type_coupling_task), pointer          :: next        => null()
   end type

   type type_coupling_task_list
      class (type_coupling_task), pointer :: first => null()
      logical                             :: includes_custom = .false.
   contains
      procedure :: remove     => coupling_task_list_remove
      procedure :: add        => coupling_task_list_add
      procedure :: add_object => coupling_task_list_add_object
   end type

   ! -----------------------------------------------------------------------
   ! Data types for variable identifiers used by biogeochemical models
   ! -----------------------------------------------------------------------

   type, abstract :: type_variable_id
      type (type_link), pointer :: link => null()
   end type

   type, extends(type_variable_id) :: type_add_id
      integer :: sum_index = -1
   end type

   type, extends(type_variable_id) :: type_horizontal_add_id
      integer :: horizontal_sum_index = -1
   end type

   type, extends(type_variable_id) :: type_dependency_id
      integer  :: index      = -1
      real(rk) :: background = 0.0_rk
   end type
```

```
195
196    type, extends(type_variable_id) :: type_horizontal_dependency_id
197       integer  :: horizontal_index = -1
198       real(rk) :: background       = 0.0_rk
199    end type
200
201    type, extends(type_horizontal_dependency_id) :: type_bottom_dependency_id
202    end type
203
204    type, extends(type_horizontal_dependency_id) :: type_surface_dependency_id
205    end type
206
207    type, extends(type_variable_id) :: type_global_dependency_id
208       integer  :: global_index = -1
209       real(rk) :: background   = 0.0_rk
210    end type
211
212    type, extends(type_dependency_id) :: type_state_variable_id
213       integer :: state_index = -1
214       type (type_add_id)            :: sms
215       type (type_add_id)            :: movement
216       type (type_horizontal_add_id) :: surface_flux
217       type (type_horizontal_add_id) :: bottom_flux
218    end type
219
220    type, extends(type_bottom_dependency_id) :: type_bottom_state_variable_id
221       integer :: bottom_state_index = -1
222       type (type_horizontal_add_id) :: bottom_sms
223    end type
224
225    type, extends(type_surface_dependency_id) :: type_surface_state_variable_id
226       integer :: surface_state_index = -1
227       type (type_horizontal_add_id) :: surface_sms
228    end type
229
230    type, extends(type_variable_id) :: type_diagnostic_variable_id
231       integer :: write_index = -1
232    end type
233
234    type, extends(type_variable_id) :: type_surface_diagnostic_variable_id
235       integer :: surface_write_index = -1
236    end type
237
238    type, extends(type_variable_id) :: type_bottom_diagnostic_variable_id
239       integer :: bottom_write_index = -1
240    end type
241
242    type, extends(type_variable_id) :: type_horizontal_diagnostic_variable_id
243       integer :: horizontal_write_index = -1
244    end type
245
246    ! ----------------------------------------------------------------------
247    ! Data types for contributions to aggregate variables.
248    ! ----------------------------------------------------------------------
249
250    type type_contribution
251       class (type_domain_specific_standard_variable), pointer :: target => null()
252       real(rk)                                        :: scale_factor = 1.0_rk
253       logical                                         :: include_background = .false.
254       type (type_contribution),            pointer :: next => null()
255    end type
256
257    type type_contribution_list
258       type (type_contribution), pointer :: first => null()
259    contains
260       procedure :: add      => contribution_list_add
261       procedure :: finalize => contribution_list_finalize
262    end type
263
264    type type_aggregate_variable_access
265       class (type_domain_specific_standard_variable), pointer :: standard_variable => null()
266       integer                                                 :: access            = access_none
267       type (type_aggregate_variable_access),          pointer :: next              => null()
268    end type
269
270    ! ----------------------------------------------------------------------
271    ! Data types for collections of variables
272    ! ----------------------------------------------------------------------
273
274    type type_link_list
275       type (type_link), pointer :: first => null()
276       type (type_link), pointer :: last  => null()
277    contains
278       procedure :: append   => link_list_append
279       procedure :: find     => link_list_find
280       procedure :: count    => link_list_count
281       procedure :: extend   => link_list_extend
282       procedure :: finalize => link_list_finalize
283    end type
284
285    type type_link_pointer
286       type (type_link),         pointer :: p    => null()
287       type (type_link_pointer), pointer :: next => null()
288    end type
289
290    type type_variable_node
291       type (type_internal_variable), pointer :: target => null()
292       type (type_variable_node),     pointer :: next   => null()
```

```
293    end type
294
295    type type_variable_set
296       type (type_variable_node), pointer :: first => null()
297    contains
298       procedure :: add      => variable_set_add
299       procedure :: update   => variable_set_update
300       procedure :: remove   => variable_set_remove
301       procedure :: contains => variable_set_contains
302       procedure :: finalize => variable_set_finalize
303    end type
304
305    type type_variable_list
306       type (type_variable_node), pointer :: first => null()
307       integer                            :: count = 0
308    contains
309       procedure :: append   => variable_list_append
310       procedure :: finalize => variable_list_finalize
311    end type
312
313    ! ------------------------------------------------------------------------
314    ! Ðata types for information on model variables and model references
315    ! ------------------------------------------------------------------------
316
317    type type_internal_variable
318       character(len=attribute_length) :: name      = ''
319       character(len=attribute_length) :: long_name = ''
320       type (type_property_dictionary) :: properties
321       character(len=attribute_length) :: units          = ''
322       real(rk)                        :: minimum        = -1.e20_rk
323       real(rk)                        :: maximum        =  1.e20_rk
324       real(rk)                        :: missing_value  = -2.e20_rk
325       real(rk)                        :: prefill_value  = -2.e20_rk
326       real(rk)                        :: initial_value  = 0.0_rk
327       integer                         :: output         = output_instantaneous
328       integer                         :: presence       = presence_internal
329       integer                         :: domain         = domain_interior
330       integer                         :: source         = source_unknown
331       integer                         :: prefill        = prefill_none
332       integer                         :: write_operator = operator_assign
333       class (type_base_model),pointer :: owner          => null()
334       type (type_contribution_list)   :: contributions
335
336       type (type_standard_variable_set) :: standard_variables
337
338       logical :: fake_state_variable = .false.
339
340       ! Only used for interior state variables:
341       logical :: no_precipitation_dilution = .false.
342       logical :: no_river_dilution         = .false.
343
344       integer, pointer :: read_index  => null()
345       integer, pointer :: write_index => null()
346       integer          :: store_index = store_index_none
347       integer          :: catalog_index = -1
348
349       ! Collections to collect information from all coupled variables.
350       type (type_integer_pointer_set)  :: read_indices, state_indices, write_indices
351       type (type_real_pointer_set)     :: background_values
352       type (type_link_list)            :: sms_list, surface_flux_list, bottom_flux_list, movement_list
353       type (type_link), pointer        :: sms_sum             => null()
354       type (type_link), pointer        :: surface_flux_sum    => null()
355       type (type_link), pointer        :: bottom_flux_sum     => null()
356       type (type_link), pointer        :: movement_sum        => null()
357       type (type_link), pointer        :: sms                 => null()
358       type (type_link), pointer        :: surface_flux        => null()
359       type (type_link), pointer        :: bottom_flux         => null()
360
361       type (type_internal_variable), pointer :: write_owner => null()
362       type (type_variable_set),      pointer :: cowriters   => null()
363       type (type_link_pointer),      pointer :: first_link  => null()
364    end type
365
366    type type_link
367       character(len=attribute_length)        :: name     = ''
368       type (type_internal_variable), pointer :: target   => null()
369       type (type_internal_variable), pointer :: original => null()
370       type (type_link), pointer              :: next     => null()
371    end type
372
373    ! ------------------------------------------------------------------------
374    ! Ðata type for custom expressions (arbitrary functions of one or more
375    ! variables).
376    ! ------------------------------------------------------------------------
377
378    type, abstract :: type_expression
379       class (type_expression), pointer :: next        => null()
380       character(len=attribute_length)  :: output_name = ''
381       integer, pointer :: out => null()
382    end type
383
384    type, abstract, extends(type_expression) :: type_interior_expression
385       !type (type_interior_data_pointer), pointer :: out => null()
386    end type
387
388    type, abstract, extends(type_expression) :: type_horizontal_expression
389       !type (type_horizontal_data_pointer), pointer :: out => null()
390    end type
```

```
391
392    ! ----------------------------------------------------------------------
393    ! Data type for collection of models
394    ! ----------------------------------------------------------------------
395
396    type type_model_list_node
397       class (type_base_model),     pointer :: model => null()
398       type (type_model_list_node), pointer :: next  => null()
399    end type
400
401    type type_model_list
402       type (type_model_list_node), pointer :: first => null()
403    contains
404       procedure :: append     => model_list_append
405       procedure :: extend     => model_list_extend
406       procedure :: find_name  => model_list_find_name
407       procedure :: find_model => model_list_find_model
408       procedure :: count      => model_list_count
409       procedure :: finalize   => model_list_finalize
410       procedure :: print      => model_list_print
411       generic   :: find       => find_name, find_model
412    end type
413
414    ! ----------------------------------------------------------------------
415    ! Base model type, used by biogeochemical models to inherit from, and by
416    ! external host to get variable lists and metadata.
417    ! ----------------------------------------------------------------------
418
419    type type_base_model
420       ! Flag determining whether the contents of the type are "frozen", i.e., they will not change anymore.
421       logical :: frozen = .false.
422
423       ! Flag determining whether this model was explicitly created by the user (by it appearing as instance in fabm.y
    aml)
424       logical :: user_created = .false.
425
426       ! Pointers to linked models in the model tree.
427       class (type_base_model), pointer :: parent => null()
428       type (type_model_list)           :: children
429
430       ! Model name and variable prefixes.
431       character(len=attribute_length) :: name      = ''
432       character(len=attribute_length) :: long_name = ''
433       character(len=attribute_length) :: type_name = ''
434
435       ! Models constituents: links to variables, coupling requests, parameters, expressions
436       type (type_link_list) :: links
437       type (type_aggregate_variable_access), pointer :: first_aggregate_variable_access => null()
438
439       type (type_hierarchical_dictionary) :: couplings
440       type (type_hierarchical_dictionary) :: parameters
441
442       class (type_expression), pointer :: first_expression => null()
443
444       type (type_coupling_task_list) :: coupling_task_list
445
446       real(rk) :: dt = 1.0_rk
447       real(rk) :: rdt__ = 1.0_rk
448
449       logical :: check_conservation = .false.
450
451       type (type_add_id)            :: extinction_id
452       type (type_horizontal_add_id) :: albedo_id
453       type (type_horizontal_add_id) :: surface_drag_id
454
455       integer, allocatable :: implemented(:)
456    contains
457
458       ! Procedure for adding child models [during initialization only]
459       procedure :: add_child
460
461       ! Procedures for adding variables [during initialization only]
462       procedure :: add_interior_variable
463       procedure :: add_horizontal_variable
464       procedure :: add_scalar_variable
465       procedure :: add_object
466
467       ! Procedures for locating links, objects, models.
468       procedure :: find_link
469       procedure :: find_object
470       procedure :: find_model
471
472       ! Procedures for requesting coupling between variables
473       procedure :: request_coupling_for_link
474       procedure :: request_coupling_for_name
475       procedure :: request_coupling_for_id
476       procedure :: request_standard_coupling_for_link
477       procedure :: request_standard_coupling_for_id
478       generic   :: request_coupling => request_coupling_for_link, request_coupling_for_name, request_coupling_for_id,
     &
479                                        request_standard_coupling_for_link, request_standard_coupling_for_id
480
481       ! Procedures that may be used to query parameter values during initialization.
482       procedure :: get_real_parameter
483       procedure :: get_integer_parameter
484       procedure :: get_logical_parameter
485       procedure :: get_string_parameter
486       generic :: get_parameter => get_real_parameter,get_integer_parameter,get_logical_parameter,get_string_parameter
```

```
487
488       procedure :: set_variable_property_real
489       procedure :: set_variable_property_integer
490       procedure :: set_variable_property_logical
491       generic   :: set_variable_property => set_variable_property_real,set_variable_property_integer,set_variable_pro
    perty_logical
492
493       procedure :: add_variable_to_aggregate_variable
494       procedure :: add_constant_to_aggregate_variable
495       generic :: add_to_aggregate_variable => add_variable_to_aggregate_variable, &
496                                               add_constant_to_aggregate_variable
497
498       ! Procedures that may be used to register model variables and dependencies during initialization.
499       procedure :: register_source
500       procedure :: register_surface_flux
501       procedure :: register_bottom_flux
502       procedure :: register_surface_source
503       procedure :: register_bottom_source
504
505       procedure :: register_interior_state_variable
506       procedure :: register_bottom_state_variable
507       procedure :: register_surface_state_variable
508
509       procedure :: register_interior_diagnostic_variable
510       procedure :: register_surface_diagnostic_variable
511       procedure :: register_bottom_diagnostic_variable
512       procedure :: register_horizontal_diagnostic_variable
513
514       procedure :: register_named_interior_dependency
515       procedure :: register_standard_interior_dependency
516       procedure :: register_universal_interior_dependency
517       procedure :: register_named_horizontal_dependency
518       procedure :: register_standard_horizontal_dependency
519       procedure :: register_standard_horizontal_dependency2
520       procedure :: register_standard_horizontal_dependency3
521       procedure :: register_universal_horizontal_dependency
522       procedure :: register_named_surface_dependency
523       procedure :: register_standard_surface_dependency
524       procedure :: register_standard_surface_dependency2
525       procedure :: register_universal_surface_dependency
526       procedure :: register_named_bottom_dependency
527       procedure :: register_standard_bottom_dependency
528       procedure :: register_standard_bottom_dependency2
529       procedure :: register_universal_bottom_dependency
530       procedure :: register_named_global_dependency
531       procedure :: register_standard_global_dependency
532
533       generic :: register_interior_dependency   => register_named_interior_dependency, register_standard_interior_dep
    endency, &
534                                                    register_universal_interior_dependency
535       generic :: register_horizontal_dependency => register_named_horizontal_dependency, register_standard_horizontal
    _dependency, &
536                                                    register_standard_horizontal_dependency2, register_standard_horizo
    ntal_dependency3, &
537                                                    register_universal_horizontal_dependency
538       generic :: register_surface_dependency    => register_named_surface_dependency, register_standard_surface_depen
    dency, &
539                                                    register_standard_surface_dependency2, register_universal_surface_
    dependency
540       generic :: register_bottom_dependency     => register_named_bottom_dependency, register_standard_bottom_depende
    ncy, &
541                                                    register_standard_bottom_dependency2, register_universal_bottom_de
    pendency
542       generic :: register_global_dependency     => register_named_global_dependency, register_standard_global_depende
    ncy
543
544       procedure :: register_interior_state_dependency
545       procedure :: register_bottom_state_dependency
546       procedure :: register_surface_state_dependency
547       procedure :: register_standard_interior_state_dependency
548       procedure :: register_standard_bottom_state_dependency
549       procedure :: register_standard_bottom_state_dependency2
550       procedure :: register_standard_surface_state_dependency
551       procedure :: register_standard_surface_state_dependency2
552
553       procedure :: register_interior_expression_dependency
554       procedure :: register_horizontal_expression_dependency
555       generic :: register_expression_dependency => register_interior_expression_dependency, register_horizontal_expre
    ssion_dependency
556
557       generic :: register_state_variable      => register_interior_state_variable, register_bottom_state_variable, &
558                                                  register_surface_state_variable
559       generic :: register_diagnostic_variable => register_interior_diagnostic_variable, register_horizontal_diagnosti
    c_variable, &
560                                                  register_surface_diagnostic_variable, register_bottom_diagnostic_var
    iable
561       generic :: register_dependency          => register_named_interior_dependency, register_standard_interior_depen
    dency, &
562                                                  register_universal_interior_dependency, &
563                                                  register_named_horizontal_dependency, register_standard_horizontal_d
    ependency, &
564                                                  register_standard_horizontal_dependency2, register_standard_horizont
    al_dependency3, &
565                                                  register_universal_horizontal_dependency, &
566                                                  register_named_surface_dependency, register_standard_surface_depende
    ncy, &
567                                                  register_standard_surface_dependency2, register_universal_surface_de
```

```
pendency, &
                                       register_named_bottom_dependency, register_standard_bottom_dependenc
y, &
                                       register_standard_bottom_dependency2, register_universal_bottom_depe
ndency, &
                                       register_named_global_dependency, register_standard_global_dependenc
y, &
                                       register_interior_expression_dependency, register_horizontal_express
ion_dependency
      generic :: register_state_dependency    => register_interior_state_dependency, register_bottom_state_dependency
, &
                                       register_surface_state_dependency, &
                                       register_standard_interior_state_dependency, &
                                       register_standard_bottom_state_dependency, &
                                       register_standard_bottom_state_dependency2, &
                                       register_standard_surface_state_dependency, &
                                       register_standard_surface_state_dependency2

      ! -----------------------------------------------------------------------------------------------------
      ! Procedures below may be overridden by biogeochemical models to provide custom data or functionality.
      ! -----------------------------------------------------------------------------------------------------

      ! Model initialization.
      procedure :: initialize               => base_initialize
      procedure :: initialize_state         => base_initialize_state
      procedure :: initialize_surface_state => base_initialize_horizontal_state
      procedure :: initialize_bottom_state  => base_initialize_horizontal_state

      ! Providing process rates and diagnostics in pelagic, at surface, and at bottom.
      procedure :: do                       => base_do
      procedure :: do_bottom                => base_do_bottom
      procedure :: do_surface               => base_do_surface
      procedure :: do_horizontal            => base_do_horizontal
      procedure :: do_ppdd                  => base_do_ppdd
      procedure :: do_bottom_ppdd           => base_do_bottom_ppdd
      procedure :: do_column                => base_do_column
      procedure :: get_vertical_movement    => base_get_vertical_movement

      ! Bookkeeping: calculate total of conserved quantities, check and repair model state.
      procedure :: check_state              => base_check_state
      procedure :: check_surface_state      => base_check_surface_state
      procedure :: check_bottom_state       => base_check_bottom_state
      procedure :: fatal_error              => base_fatal_error
      procedure :: log_message              => base_log_message
      procedure :: get_path                 => base_get_path

      ! Hooks called by FABM - usable by inheriting models
      procedure :: before_coupling => base_before_coupling
      procedure :: after_coupling  => base_after_coupling

      procedure :: implements
      procedure :: register_implemented_routines

      procedure :: finalize => base_finalize

      ! Deprecated as of FABM 1.0
      procedure :: get_light                => base_get_light
      procedure :: get_light_extinction     => base_get_light_extinction
      procedure :: get_drag                 => base_get_drag
      procedure :: get_albedo               => base_get_albedo
   end type type_base_model

   ! ======================================================================================================
   ! Derived type for cache for all input/output during model calls.
   ! ======================================================================================================

   type type_cache
      ! Number of active items in a single cache line [first dimension of any spatially explicit caches below]
      integer :: n = 1

      ! Read cache (separate interior, horizontal, scalar fields).
      real(rk), allocatable _DIMENSION_SLICE_PLUS_1_            :: read
      real(rk), allocatable _DIMENSION_HORIZONTAL_SLICE_PLUS_1_ :: read_hz
      real(rk), allocatable, dimension(:)                       :: read_scalar

#ifdef _FABM_MASK_TYPE_
      ! Index mapping between source arrays and packed data
      integer, allocatable _DIMENSION_SLICE_ :: ipack
      integer, allocatable _DIMENSION_SLICE_ :: iunpack
#endif

      logical :: repair
      logical :: valid
      logical :: set_interior
      logical :: set_horizontal
   end type

   type, extends(type_cache) :: type_interior_cache
      ! Write cache (separate interior, horizontal fields).
      real(rk), allocatable _DIMENSION_SLICE_PLUS_1_  :: write
   end type

   type, extends(type_cache) :: type_horizontal_cache
      ! Write cache (separate interior, horizontal fields).
      real(rk), allocatable _DIMENSION_HORIZONTAL_SLICE_PLUS_1_ :: write_hz
   end type

   type, extends(type_cache) :: type_vertical_cache
```

```
660       ! Write cache (separate interior, horizontal fields).
661       real(rk), allocatable _DIMENSION_SLICE_PLUS_1_            :: write
662       real(rk), allocatable _DIMENSION_HORIZONTAL_SLICE_PLUS_1_ :: write_hz
663    end type
664
665    ! ================================================================================================================
666    ! Base type for a model object factory (generates a model object from a model name)
667    ! An implementation of this type is provided in fabm_library.F90.
668    ! Institutes or groups can create inherit from this type to create their own model factories,
669    ! which then need to be added to the root factory in fabm_library.F90.
670    ! This makes it possible to introduce a large number of new models with only two lines added
671    ! in the FABM core.
672    ! ================================================================================================================
673
674    type, public :: type_version
675       character(len=attribute_length) :: module_name    = ''
676       character(len=attribute_length) :: version_string = ''
677       type (type_version), pointer    :: next           => null()
678    end type
679    type (type_version), pointer, save, public :: first_module_version => null()
680
681    type type_base_model_factory_node
682       character(len=attribute_length)              :: prefix  = ''
683       class (type_base_model_factory),     pointer :: factory => null()
684       type (type_base_model_factory_node), pointer :: next    => null()
685    end type
686
687    type, public :: type_base_model_factory
688       type (type_base_model_factory_node), pointer :: first_child => null()
689       logical                                      :: initialized = .false.
690    contains
691       procedure :: initialize       => abstract_model_factory_initialize
692       procedure :: add              => abstract_model_factory_add
693       procedure :: create           => abstract_model_factory_create
694       procedure :: register_version => abstract_model_factory_register_version
695       procedure :: finalize         => abstract_model_factory_finalize
696    end type
697
698    class (type_base_model_factory), pointer, save, public :: factory => null()
699
700 contains
701
702    subroutine base_initialize(self, configunit)
703       class (type_base_model), intent(inout), target :: self
704       integer,                 intent(in)            :: configunit
705    end subroutine
706
707    subroutine base_initialize_state(self, _ARGUMENTS_INITIALIZE_STATE_)
708       class (type_base_model), intent(in) :: self
709       _DECLARE_ARGUMENTS_INITIALIZE_STATE_
710    end subroutine
711
712    subroutine base_initialize_horizontal_state(self, _ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_)
713       class (type_base_model), intent(in) :: self
714       _DECLARE_ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_
715    end subroutine
716
717    ! Providing process rates and diagnostics
718    subroutine base_do(self, _ARGUMENTS_DO_)
719       class (type_base_model), intent(in) :: self
720       _DECLARE_ARGUMENTS_DO_
721    end subroutine
722
723    subroutine base_do_ppdd(self, _ARGUMENTS_DO_PPDD_)
724       class (type_base_model), intent(in) :: self
725       _DECLARE_ARGUMENTS_DO_PPDD_
726       call self%do(_ARGUMENTS_DO_)
727    end subroutine
728
729    subroutine base_do_bottom(self, _ARGUMENTS_DO_BOTTOM_)
730       class (type_base_model), intent(in) :: self
731       _DECLARE_ARGUMENTS_DO_BOTTOM_
732    end subroutine
733
734    subroutine base_do_bottom_ppdd(self, _ARGUMENTS_DO_BOTTOM_PPDD_)
735       class (type_base_model), intent(in) :: self
736       _DECLARE_ARGUMENTS_DO_BOTTOM_PPDD_
737    end subroutine
738
739    subroutine base_do_surface(self, _ARGUMENTS_DO_SURFACE_)
740       class (type_base_model), intent(in) :: self
741       _DECLARE_ARGUMENTS_DO_SURFACE_
742    end subroutine
743
744    subroutine base_do_horizontal(self, _ARGUMENTS_HORIZONTAL_)
745       class (type_base_model), intent(in) :: self
746       _DECLARE_ARGUMENTS_HORIZONTAL_
747    end subroutine
748
749    subroutine base_do_column(self, _ARGUMENTS_DO_COLUMN_)
750       class (type_base_model), intent(in) :: self
751       _DECLARE_ARGUMENTS_DO_COLUMN_
752       call self%get_light(_ARGUMENTS_DO_COLUMN_)
753    end subroutine
754
755    subroutine base_get_vertical_movement(self, _ARGUMENTS_GET_VERTICAL_MOVEMENT_)
756       class (type_base_model), intent(in) :: self
757       _DECLARE_ARGUMENTS_GET_VERTICAL_MOVEMENT_
```

```
   end subroutine

   subroutine base_check_state(self, _ARGUMENTS_CHECK_STATE_)
      class (type_base_model), intent(in) :: self
      _DECLARE_ARGUMENTS_CHECK_STATE_
   end subroutine

   subroutine base_check_surface_state(self, _ARGUMENTS_CHECK_SURFACE_STATE_)
      class (type_base_model), intent(in) :: self
      _DECLARE_ARGUMENTS_CHECK_SURFACE_STATE_
   end subroutine

   subroutine base_check_bottom_state(self, _ARGUMENTS_CHECK_BOTTOM_STATE_)
      class (type_base_model), intent(in) :: self
      _DECLARE_ARGUMENTS_CHECK_BOTTOM_STATE_
   end subroutine

   recursive subroutine base_finalize(self)
      class (type_base_model), intent(inout) :: self

      type (type_model_list_node),           pointer :: node
      type (type_aggregate_variable_access), pointer :: aggregate_variable_access, next_aggregate_variable_access
      class (type_expression),               pointer :: expression, next_expression
      type (type_link),                      pointer :: link

      node => self%children%first
      do while (associated(node))
         call node%model%finalize()
         deallocate(node%model)
         node => node%next
      end do
      call self%children%finalize()

      call self%couplings%finalize()
      call self%parameters%finalize()

      aggregate_variable_access => self%first_aggregate_variable_access
      do while (associated(aggregate_variable_access))
         next_aggregate_variable_access => aggregate_variable_access%next
         deallocate(aggregate_variable_access)
         aggregate_variable_access => next_aggregate_variable_access
      end do
      self%first_aggregate_variable_access => null()

      expression => self%first_expression
      do while (associated(expression))
         next_expression => expression%next
         deallocate(expression)
         expression => next_expression
      end do
      self%first_expression => null()

      link => self%links%first
      do while (associated(link))
         if (index(link%name, '/') == 0) then
            call finalize_variable(link%original)
            deallocate(link%original)
         end if
         link => link%next
      end do
      call self%links%finalize()

   contains

      subroutine finalize_variable(variable)
         type (type_internal_variable), intent(inout) :: variable

         type (type_link_pointer), pointer :: link_pointer, next_link_pointer

         call variable%standard_variables%finalize()
         call variable%contributions%finalize()
         call variable%read_indices%finalize()
         call variable%state_indices%finalize()
         call variable%write_indices%finalize()
         call variable%sms_list%finalize()
         call variable%surface_flux_list%finalize()
         call variable%bottom_flux_list%finalize()
         call variable%movement_list%finalize()
         if (associated(variable%cowriters)) then
            call variable%cowriters%finalize()
            deallocate(variable%cowriters)
         end if
         link_pointer => variable%first_link
         do while (associated(link_pointer))
            next_link_pointer => link_pointer%next
            deallocate(link_pointer)
            link_pointer => next_link_pointer
         end do
      end subroutine

   end subroutine

   ! Deprecated as of FABM 1.0:

   subroutine base_get_light_extinction(self, _ARGUMENTS_GET_EXTINCTION_)
      class (type_base_model), intent(in) :: self
      _DECLARE_ARGUMENTS_GET_EXTINCTION_
   end subroutine
```

```
856
857    subroutine base_get_drag(self, _ARGUMENTS_GET_DRAG_)
858       class (type_base_model), intent(in) :: self
859       _DECLARE_ARGUMENTS_GET_DRAG_
860    end subroutine
861
862    subroutine base_get_albedo(self, _ARGUMENTS_GET_ALBEDO_)
863       class (type_base_model), intent(in) :: self
864       _DECLARE_ARGUMENTS_GET_ALBEDO_
865    end subroutine
866
867    subroutine base_get_light(self, _ARGUMENTS_DO_COLUMN_)
868       class (type_base_model), intent(in) :: self
869       _DECLARE_ARGUMENTS_DO_COLUMN_
870    end subroutine
871
872    function base_get_path(self) result(path)
873       class (type_base_model), intent(in), target :: self
874       character(len=attribute_length)             :: path
875
876       class (type_base_model), pointer :: current
877
878       path = ''
879       current => self
880       do while (associated(current%parent))
881          path = '/' // trim(current%name) // trim(path)
882          current => current%parent
883       end do
884    end function
885
886    subroutine base_fatal_error(self, location, message)
887       class (type_base_model), intent(in) :: self
888       character(len=*),        intent(in) :: location, message
889       if (self%name /= '') then
890          call driver%fatal_error('model ' // trim(self%get_path()) // ', ' // trim(location), message)
891       else
892          call driver%fatal_error(location, message)
893       end if
894    end subroutine
895
896    subroutine base_log_message(self, message)
897       class (type_base_model), intent(in) :: self
898       character(len=*),        intent(in) :: message
899       if (self%name /= '') then
900          call driver%log_message('model "' // trim(self%name) // '": ' // message)
901       else
902          call driver%log_message(message)
903       end if
904    end subroutine
905
906    subroutine base_before_coupling(self)
907       class (type_base_model), intent(inout) :: self
908    end subroutine
909
910    subroutine base_after_coupling(self)
911       class (type_base_model), intent(inout) :: self
912    end subroutine
913
914    function implements(self, source) result(is_implemented)
915       class (type_base_model), intent(in) :: self
916       integer,                 intent(in) :: source
917       logical                             :: is_implemented
918
919       integer :: i
920
921       is_implemented = .true.
922       if (allocated(self%implemented)) then
923          do i = 1, size(self%implemented)
924             if (self%implemented(i) == source) return
925          end do
926          is_implemented = .false.
927       end if
928    end function
929
930    subroutine register_implemented_routines(self, sources)
931       class (type_base_model), intent(inout) :: self
932       integer, optional,       intent(in)    :: sources(:)
933       if (allocated(self%implemented)) deallocate(self%implemented)
934       if (present(sources)) then
935          allocate(self%implemented(size(sources)))
936          self%implemented(:) = sources
937       else
938          allocate(self%implemented(0))
939       end if
940    end subroutine
941
942    recursive subroutine add_child(self, model, name, long_name, configunit)
943       class (type_base_model), target, intent(inout) :: self, model
944       character(len=*),                intent(in)    :: name
945       character(len=*), optional,      intent(in)    :: long_name
946       integer, optional,               intent(in)    :: configunit
947
948       integer                              :: islash
949       class (type_base_model),     pointer :: parent
950       type (type_model_list_node), pointer :: child
951       integer                              :: ind
952
953       ! If a path with / is given, redirect to tentative parent model.
```

```
954       islash = index(name, '/', .true.)
955       if (islash /= 0) then
956          parent => self%find_model(name(:islash - 1))
957          if (.not. associated(parent)) call self%fatal_error('add_child', &
958             'Proposed parent model "' // trim(name(:islash - 1)) // '" was not found.')
959          call parent%add_child(model, name(islash + 1:), long_name, configunit)
960          return
961       end if
962
963       if (associated(model%parent)) call self%fatal_error('add_child', &
964          'The provided child model "' // trim(name) // '" has already been assigned parent ' // trim(model%parent%nam
    e) // '.')
965
966       if (name == '*') then
967          ! This instance is for internal use only - auto-generate a unique name
968          ind = 1
969          do
970             write (model%name, '("_", i0)') ind
971             child => self%children%first
972             do while (associated(child))
973                if (child%model%name == model%name) exit
974                child => child%next
975             end do
976             if (.not. associated(child)) exit
977             ind = ind + 1
978          end do
979       else
980          ! Ascertain whether the provided name is valid.
981          if (name == '') call self%fatal_error('add_child', 'Invalid model name "' // trim(name) // &
982             '". Names cannot be empty.')
983          if (name(1:1) == '_') call self%fatal_error('add_child', 'Invalid model name "' // trim(name) // &
984             '". Names beginning with underscore are reserved for internal use.')
985          if (len_trim(name) > len(model%name)) call self%fatal_error('add_child', 'Invalid model name "' // trim(name
    ) // &
986             '". This name is longer than the maximum allowed number of characters.')
987          if (name /= get_safe_name(name)) call self%fatal_error('add_child', 'Invalid model name " '// trim(name) //
    &
988             '". Names can contain letters, digits and underscores only.')
989
990          ! Make sure a child with this name does not exist yet.
991          child => self%children%first
992          do while (associated(child))
993             if (child%model%name == name) call self%fatal_error('add_child', &
994                'A child model with name "' // trim(name) // '" already exists.')
995             child => child%next
996          end do
997          model%name = name
998       end if
999
1000      if (present(long_name)) then
1001         model%long_name = trim(long_name)
1002      else
1003         model%long_name = trim(model%name)
1004      end if
1005      model%parent => self
1006      call self%parameters%add_child(model%parameters, trim(model%name))
1007      call self%couplings%add_child(model%couplings, trim(model%name))
1008      call self%children%append(model)
1009      call model%initialize(-1)
1010      model%rdt__ = 1._rk / model%dt
1011
1012      if (model%implements(source_get_light_extinction)) then
1013         call model%add_interior_variable('_attenuation_coefficient_of_photosynthetic_radiative_flux', 'm-1', &
1014            'light extinction contribution computed by get_light_extinction', fill_value=0.0_rk, missing_value=0.0_rk
    , &
1015            output=output_none, write_index=model%extinction_id%sum_index, link=model%extinction_id%link, &
1016            source=source_get_light_extinction)
1017         model%extinction_id%link%target%write_operator = operator_add
1018         call model%add_to_aggregate_variable(standard_variables%attenuation_coefficient_of_photosynthetic_radiative_
    flux, &
1019            model%extinction_id)
1020      end if
1021
1022      if (model%implements(source_get_albedo)) then
1023         call model%add_horizontal_variable('_surface_albedo', '-', &
1024            'albedo contribution computed by get_albedo', fill_value=0.0_rk, missing_value=0.0_rk, &
1025            output=output_none, write_index=model%albedo_id%horizontal_sum_index, link=model%albedo_id%link, &
1026            source=source_get_albedo)
1027         model%albedo_id%link%target%write_operator = operator_add
1028         call model%add_to_aggregate_variable(standard_variables%surface_albedo, model%albedo_id)
1029      end if
1030
1031      if (model%implements(source_get_drag)) then
1032         call model%add_horizontal_variable('_surface_drag_coefficient_in_air', '-', &
1033            'surface drag contribution computed by get_drag', fill_value=0.0_rk, missing_value=0.0_rk, &
1034            output=output_none, write_index=model%surface_drag_id%horizontal_sum_index, link=model%surface_drag_id%li
    nk, &
1035            source=source_get_drag)
1036         model%surface_drag_id%link%target%write_operator = operator_add
1037         call model%add_to_aggregate_variable(standard_variables%surface_drag_coefficient_in_air, model%surface_drag_
    id)
1038      end if
1039   end subroutine add_child
1040
1041   subroutine set_variable_property_real(self, variable, name, value)
1042      class (type_base_model),  intent(inout) :: self
1043      class (type_variable_id), intent(inout) :: variable
1044      character(len=*),         intent(in)    :: name
```

```
1045      real(rk),                intent(in)    :: value
1046      if (.not. associated(variable%link)) call self%fatal_error('set_variable_property_real', 'variable has not been
 registered')
1047      call variable%link%target%properties%set_real(name, value)
1048   end subroutine
1049
1050   subroutine set_variable_property_integer(self, variable, name, value)
1051      class (type_base_model),  intent(inout) :: self
1052      class (type_variable_id), intent(inout) :: variable
1053      character(len=*),         intent(in)    :: name
1054      integer,                  intent(in)    :: value
1055      if (.not. associated(variable%link)) call self%fatal_error('set_variable_property_integer', 'variable has not b
een registered')
1056      call variable%link%target%properties%set_integer(name, value)
1057   end subroutine
1058
1059   subroutine set_variable_property_logical(self, variable, name, value)
1060      class (type_base_model), intent(inout) :: self
1061      class (type_variable_id),intent(inout) :: variable
1062      character(len=*),        intent(in)    :: name
1063      logical,                 intent(in)    :: value
1064      if (.not.associated(variable%link)) call self%fatal_error('set_variable_property_logical', 'variable has not be
en registered')
1065      call variable%link%target%properties%set_logical(name, value)
1066   end subroutine
1067
1068   subroutine add_variable_to_aggregate_variable(self, target, variable_id, scale_factor, include_background)
1069      use fabm_standard_variables   ! workaround for bug in Cray compiler 8.3.7
1070      class (type_base_model),             intent(inout) :: self
1071      class (type_base_standard_variable), intent(in)    :: target
1072      class (type_variable_id),            intent(inout) :: variable_id
1073      real(rk), optional,                  intent(in)    :: scale_factor
1074      logical, optional,                   intent(in)    :: include_background
1075
1076      class (type_base_standard_variable), pointer :: standard_variable
1077
1078      if (.not. target%aggregate_variable) call self%fatal_error('add_variable_to_aggregate_variable', &
1079           'target "' // trim(target%name) // '" is not an aggregate variable.')
1080      if (.not. associated(variable_id%link)) call self%fatal_error('add_to_aggregate_variable', &
1081         'variable added to ' // trim(target%name) // ' has not been registered')
1082      standard_variable => target%resolve()
1083      select type (standard_variable)
1084      class is (type_universal_standard_variable)
1085         select case(variable_id%link%target%domain)
1086         case (domain_interior)
1087            call variable_id%link%target%contributions%add(standard_variable%in_interior(), scale_factor, include_bac
kground)
1088         case (domain_horizontal)
1089            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_b
ackground)
1090         case (domain_surface)
1091            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_b
ackground)
1092            call variable_id%link%target%contributions%add(standard_variable%at_surface(), scale_factor, include_back
ground)
1093         case (domain_bottom)
1094            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_b
ackground)
1095            call variable_id%link%target%contributions%add(standard_variable%at_bottom(), scale_factor, include_backg
round)
1096         end select
1097      class is (type_domain_specific_standard_variable)
1098         call variable_id%link%target%contributions%add(standard_variable, scale_factor, include_background)
1099      end select
1100   end subroutine add_variable_to_aggregate_variable
1101
1102   subroutine add_constant_to_aggregate_variable(self, target, value)
1103      class (type_base_model),                        intent(inout) :: self
1104      class (type_domain_specific_standard_variable), intent(in)    :: target
1105      real(rk),                                       intent(in)    :: value
1106
1107      class (type_domain_specific_standard_variable), pointer :: standard_variable
1108      type (type_link),                               pointer :: link
1109
1110      if (.not. target%aggregate_variable) call self%fatal_error('add_constant_to_aggregate_variable', &
1111           'target "' // trim(target%name) // '" is not an aggregate variable.')
1112      standard_variable => target%typed_resolve()
1113
1114      link => null()
1115      select type (standard_variable)
1116      class is (type_interior_standard_variable)
1117         call self%add_interior_variable('_constant_*', standard_variable%units, standard_variable%name, source=sourc
e_constant, &
1118              fill_value=value, output=output_none, link=link)
1119         call link%target%contributions%add(standard_variable)
1120      class is (type_surface_standard_variable)
1121         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=sou
rce_constant, &
1122              fill_value=value, domain=domain_surface, output=output_none, link=link)
1123      class is (type_bottom_standard_variable)
1124         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=sou
rce_constant, &
1125              fill_value=value, domain=domain_bottom, output=output_none, link=link)
1126      class is (type_horizontal_standard_variable)
1127         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=sou
rce_constant, &
1128              fill_value=value, output=output_none, link=link)
1129      end select
```

```
1130        call link%target%contributions%add(standard_variable)
1131     end subroutine add_constant_to_aggregate_variable
1132
1133     subroutine contribution_list_add(self, standard_variable, scale_factor, include_background)
1134        class (type_contribution_list),      intent(inout) :: self
1135        class (type_domain_specific_standard_variable), target :: standard_variable
1136        real(rk), optional,                  intent(in)    :: scale_factor
1137        logical, optional,                   intent(in)    :: include_background
1138
1139        type (type_contribution), pointer :: contribution
1140        logical,                  pointer :: pmember
1141
1142        ! If the scale factor is 0, no need to register any contribution.
1143        if (present(scale_factor)) then
1144           if (scale_factor == 0.0_rk) return
1145        end if
1146
1147        ! First look for existing contribution to this aggregate variable.
1148        contribution => self%first
1149        pmember => standard_variable%aggregate_variable
1150        do while (associated(contribution))
1151           ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
     erences.
1152           if (associated(pmember, contribution%target%aggregate_variable)) exit
1153           contribution => contribution%next
1154        end do
1155
1156        if (.not. associated(contribution)) then
1157           ! No contribution to this aggregate variable exists - prepend it to the list.
1158           allocate(contribution)
1159           contribution%next => self%first
1160           self%first => contribution
1161        end if
1162
1163        ! Store contribution attributes
1164        contribution%target => standard_variable
1165        if (present(scale_factor)) contribution%scale_factor = scale_factor
1166        if (present(include_background)) contribution%include_background = include_background
1167     end subroutine
1168
1169     subroutine contribution_list_finalize(self)
1170        class (type_contribution_list), intent(inout) :: self
1171
1172        type (type_contribution), pointer :: contribution, next_contribution
1173
1174        contribution => self%first
1175        do while (associated(contribution))
1176           next_contribution => contribution%next
1177           deallocate(contribution)
1178           contribution => next_contribution
1179        end do
1180        self%first => null()
1181     end subroutine
1182
1183     subroutine model_list_append(self, model)
1184        class (type_model_list), intent(inout) :: self
1185        class (type_base_model), target        :: model
1186
1187        type (type_model_list_node), pointer :: node
1188
1189        if (.not. associated(self%first)) then
1190           allocate(self%first)
1191           node => self%first
1192        else
1193           node => self%first
1194           do while (associated(node%next))
1195              node => node%next
1196           end do
1197           allocate(node%next)
1198           node => node%next
1199        end if
1200        node%model => model
1201     end subroutine
1202
1203     subroutine model_list_extend(self, source)
1204        class (type_model_list), intent(inout) :: self
1205        class (type_model_list), intent(in)    :: source
1206
1207        type (type_model_list_node), pointer :: node
1208
1209        node => source%first
1210        do while (associated(node))
1211           call self%append(node%model)
1212           node => node%next
1213        end do
1214     end subroutine
1215
1216     function model_list_find_name(self, name) result(node)
1217        class (type_model_list), intent(in) :: self
1218        character(len=*),        intent(in) :: name
1219
1220        type (type_model_list_node), pointer :: node
1221
1222        node => self%first
1223        do while (associated(node))
1224           if (node%model%name == name) return
1225           node => node%next
1226        end do
```

```
1227    end function model_list_find_name
1228
1229    function model_list_find_model(self, model) result(node)
1230       class (type_model_list),         intent(in) :: self
1231       class (type_base_model), target, intent(in) :: model
1232
1233       type (type_model_list_node), pointer :: node
1234       logical,                     pointer :: pmember
1235
1236       node => self%first
1237       pmember => model%frozen
1238       do while (associated(node))
1239          ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
    erences.
1240          if (associated(pmember, node%model%frozen)) return
1241          node => node%next
1242       end do
1243    end function model_list_find_model
1244
1245    subroutine model_list_print(self)
1246       class (type_model_list),       intent(in) :: self
1247
1248       type (type_model_list_node),pointer :: node
1249
1250       node => self%first
1251       do while (associated(node))
1252          call driver%log_message(node%model%get_path())
1253          node => node%next
1254       end do
1255    end subroutine
1256
1257    function model_list_count(self, model) result(count)
1258       class (type_model_list),         intent(in) :: self
1259       class (type_base_model), target, intent(in) :: model
1260
1261       integer :: count
1262
1263       type (type_model_list_node), pointer :: node
1264       logical,                     pointer :: pmember
1265
1266       count = 0
1267       node => self%first
1268       pmember => model%frozen
1269       do while (associated(node))
1270          ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
    erences.
1271          if (associated(pmember, node%model%frozen)) count = count + 1
1272          node => node%next
1273       end do
1274    end function
1275
1276    subroutine model_list_finalize(self)
1277       class (type_model_list), intent(inout) :: self
1278
1279       type (type_model_list_node), pointer :: node, next
1280
1281       node => self%first
1282       do while (associated(node))
1283          next => node%next
1284          deallocate(node)
1285          node => next
1286       end do
1287       self%first => null()
1288    end subroutine
1289
1290    function link_list_find(self, name) result(link)
1291       class (type_link_list), intent(in) :: self
1292       character(len=*),       intent(in) :: name
1293
1294       type (type_link), pointer :: link
1295
1296       link => self%first
1297       do while (associated(link))
1298          if (link%name == name) return
1299          link => link%next
1300       end do
1301    end function link_list_find
1302
1303    function link_list_append(self, target, name) result(link)
1304       class (type_link_list),  intent(inout) :: self
1305       type (type_internal_variable), pointer :: target
1306       character(len=*),        intent(in)    :: name
1307
1308       type (type_link), pointer :: link
1309
1310       ! Append a new link to the list.
1311       if (.not. associated(self%first)) then
1312          allocate(self%first)
1313          self%last => self%first
1314       else
1315          allocate(self%last%next)
1316          self%last => self%last%next
1317       end if
1318
1319       ! Set link attributes.
1320       link => self%last
1321       link%name = name
1322       link%target => target
```

```
1323        link%original => target
1324     end function link_list_append
1325
1326     subroutine link_list_extend(self, source)
1327        class (type_link_list), intent(inout) :: self
1328        class (type_link_list), intent(in)    :: source
1329
1330        type (type_link), pointer :: source_link, link
1331
1332        source_link => source%first
1333        do while (associated(source_link))
1334           link => self%append(source_link%target, source_link%name)
1335           source_link => source_link%next
1336        end do
1337     end subroutine link_list_extend
1338
1339     function link_list_count(self) result(count)
1340        class (type_link_list), intent(in) :: self
1341        integer                            :: count
1342
1343        type (type_link), pointer :: link
1344
1345        count = 0
1346        link => self%first
1347        do while (associated(link))
1348           count = count + 1
1349           link => link%next
1350        end do
1351     end function link_list_count
1352
1353     subroutine link_list_finalize(self)
1354        class (type_link_list), intent(inout) :: self
1355
1356        type (type_link), pointer :: link, next
1357
1358        link => self%first
1359        do while (associated(link))
1360           next => link%next
1361           deallocate(link)
1362           link => next
1363        end do
1364        self%first => null()
1365     end subroutine link_list_finalize
1366
1367     subroutine create_coupling_task(self, link, task)
1368        class (type_base_model),  intent(inout) :: self
1369        type (type_link), target, intent(inout) :: link
1370        class (type_coupling_task), pointer     :: task
1371
1372        type (type_link), pointer :: current_link
1373
1374        ! First make sure that we are called for a link that we own ourselves.
1375        current_link => self%links%first
1376        do while (associated(current_link))
1377           if (associated(current_link, link)) exit
1378           current_link => current_link%next
1379        end do
1380        if (.not.associated(current_link)) call self%fatal_error('request_coupling_for_link', &
1381           'Couplings can only be requested for variables that you own yourself.')
1382
1383        ! Make sure that the link also points to a variable that we registered ourselves,
1384        ! rather than one registered by a child model.
1385        if (index(link%name, '/') /= 0) call self%fatal_error('request_coupling_for_link', &
1386           'Couplings can only be requested for variables that you registered yourself, &
1387           &not inherited ones such as the current ' // trim(link%name) // '.')
1388
1389        ! Create a coupling task (reuse existing one if available, and not user-specified)
1390        call self%coupling_task_list%add(link, .false., task)
1391     end subroutine create_coupling_task
1392
1393     subroutine request_coupling_for_link(self, link, master)
1394        class (type_base_model),  intent(inout) :: self
1395        type (type_link), target, intent(inout) :: link
1396        character(len=*),         intent(in)    :: master
1397
1398        class (type_coupling_task), pointer :: task
1399
1400        ! Create a coupling task (reuse existing one if available, and not user-specified)
1401        call create_coupling_task(self, link, task)
1402        if (.not. associated(task)) return   ! We already have a user-specified task, which takes priority
1403
1404        ! Configure coupling task
1405        task%master_name = master
1406     end subroutine request_coupling_for_link
1407
1408     recursive subroutine request_coupling_for_name(self, slave, master)
1409        class (type_base_model), intent(inout), target :: self
1410        character(len=*),        intent(in)            :: slave, master
1411
1412        class (type_base_model), pointer :: parent
1413        type (type_link),        pointer :: link
1414        integer                          :: islash
1415
1416        ! If a path with / is given, redirect to tentative parent model.
1417        islash = index(slave, '/', .true.)
1418        if (islash /= 0) then
1419           parent => self%find_model(slave(:islash - 1))
1420           call request_coupling_for_name(parent, slave(islash + 1:), master)
```

```
1421          return
1422       end if
1423
1424       link => self%links%find(slave)
1425       if (.not. associated(link)) call self%fatal_error('request_coupling_for_name', &
1426          'Specified slave (' // trim(slave) // ') not found. Make sure the variable is registered before calling requ
     est_coupling.')
1427       call request_coupling_for_link(self, link, master)
1428    end subroutine request_coupling_for_name
1429
1430    subroutine request_coupling_for_id(self, id, master)
1431       class (type_base_model),  intent(inout) :: self
1432       class (type_variable_id), intent(inout) :: id
1433       character(len=*),         intent(in)    :: master
1434
1435       if (.not. associated(id%link)) call self%fatal_error('request_coupling_for_id', &
1436          'The provided variable identifier has not been registered yet.')
1437       call self%request_coupling(id%link, master)
1438    end subroutine request_coupling_for_id
1439
1440    subroutine request_standard_coupling_for_link(self, link, master)
1441       use fabm_standard_variables   ! workaround for bug in Cray compiler 8.3.4
1442       class (type_base_model),                        intent(inout)      :: self
1443       type (type_link), target,                       intent(inout)      :: link
1444       class (type_domain_specific_standard_variable), intent(in), target :: master
1445
1446       class (type_coupling_task), pointer :: task
1447
1448       call create_coupling_task(self, link, task)
1449       if (.not. associated(task)) return   ! We already have a user-specified task, which takes priority
1450       task%master_standard_variable => master%typed_resolve()
1451    end subroutine request_standard_coupling_for_link
1452
1453    subroutine request_standard_coupling_for_id(self, id, master)
1454       class (type_base_model),                        intent(inout)      :: self
1455       class (type_variable_id),                       intent(inout)      :: id
1456       class (type_domain_specific_standard_variable), intent(in), target :: master
1457
1458       if (.not. associated(id%link)) call self%fatal_error('request_standard_coupling_for_id', &
1459          'The provided variable identifier has not been registered yet.')
1460       call self%request_standard_coupling_for_link(id%link, master)
1461    end subroutine request_standard_coupling_for_id
1462
1463    subroutine integer_pointer_set_append(self, value)
1464       class (type_integer_pointer_set), intent(inout) :: self
1465       integer, target                                 :: value
1466
1467       type (type_integer_pointer), allocatable :: oldarray(:)
1468
1469       ! Create a new list of integer pointers, or extend it if already allocated.
1470       if (.not. allocated(self%pointers)) then
1471          allocate(self%pointers(1))
1472       else
1473          call move_alloc(self%pointers, oldarray)
1474          allocate(self%pointers(size(oldarray) + 1))
1475          self%pointers(1:size(oldarray)) = oldarray
1476          deallocate(oldarray)
1477       end if
1478
1479       ! Add pointer to provided integer to the list.
1480       self%pointers(size(self%pointers))%p => value
1481       self%pointers(size(self%pointers))%p = self%value
1482    end subroutine integer_pointer_set_append
1483
1484    subroutine integer_pointer_set_extend(self, other)
1485       class (type_integer_pointer_set), intent(inout) :: self
1486       class (type_integer_pointer_set), intent(in)    :: other
1487
1488       integer :: i
1489
1490       if (allocated(other%pointers)) then
1491          do i=1,size(other%pointers)
1492             call self%append(other%pointers(i)%p)
1493          end do
1494       end if
1495    end subroutine integer_pointer_set_extend
1496
1497    subroutine integer_pointer_set_finalize(self)
1498       class (type_integer_pointer_set), intent(inout) :: self
1499
1500       if (allocated(self%pointers)) deallocate(self%pointers)
1501       self%value = -1
1502    end subroutine integer_pointer_set_finalize
1503
1504    subroutine integer_pointer_set_set_value(self, value)
1505       class (type_integer_pointer_set), intent(inout) :: self
1506       integer,                          intent(in)    :: value
1507
1508       integer :: i
1509
1510       if (allocated(self%pointers)) then
1511          do i=1,size(self%pointers)
1512             self%pointers(i)%p = value
1513          end do
1514       end if
1515       self%value = value
1516    end subroutine integer_pointer_set_set_value
1517
```

```
   logical function integer_pointer_set_is_empty(self)
      class (type_integer_pointer_set), intent(in) :: self

      integer_pointer_set_is_empty = .not. allocated(self%pointers)
   end function integer_pointer_set_is_empty

   subroutine real_pointer_set_append(self, value)
      class (type_real_pointer_set), intent(inout) :: self
      real(rk),target                              :: value

      type (type_real_pointer), allocatable :: oldarray(:)

      ! Create a new list of real pointers, or extend it if already allocated.
      if (.not. allocated(self%pointers)) then
         allocate(self%pointers(1))
      else
         call move_alloc(self%pointers, oldarray)
         allocate(self%pointers(size(oldarray) + 1))
         self%pointers(1:size(oldarray)) = oldarray
         deallocate(oldarray)
      end if

      ! Add pointer to provided real to the list.
      self%pointers(size(self%pointers))%p => value
      self%pointers(size(self%pointers))%p = self%pointers(1)%p
   end subroutine real_pointer_set_append

   subroutine real_pointer_set_extend(self, other)
      class (type_real_pointer_set), intent(inout) :: self
      class (type_real_pointer_set), intent(in)    :: other

      integer :: i

      if (allocated(other%pointers)) then
         do i=1,size(other%pointers)
            call self%append(other%pointers(i)%p)
         end do
      end if
   end subroutine real_pointer_set_extend

   subroutine real_pointer_set_set_value(self, value)
      class (type_real_pointer_set), intent(inout) :: self
      real(rk),                      intent(in)    :: value

      integer :: i

      if (allocated(self%pointers)) then
         do i=1,size(self%pointers)
            self%pointers(i)%p = value
         end do
      end if
   end subroutine real_pointer_set_set_value

   subroutine register_interior_state_variable(self, id, name, units, long_name, &
                                               initial_value, vertical_movement, specific_light_extinction, &
                                               minimum, maximum, missing_value, &
                                               no_precipitation_dilution, no_river_dilution, &
                                               standard_variable, presence, background_value)
      class (type_base_model),             intent(inout)         :: self
      type (type_state_variable_id),       intent(inout), target :: id
      character(len=*),                    intent(in)            :: name, long_name, units
      real(rk),                            intent(in), optional  :: initial_value,vertical_movement,specific_light_ex
tinction
      real(rk),                            intent(in), optional  :: minimum, maximum,missing_value,background_value
      logical,                             intent(in), optional  :: no_precipitation_dilution,no_river_dilution
      class (type_base_standard_variable), intent(in), optional  :: standard_variable
      integer,                             intent(in), optional  :: presence

      call self%add_interior_variable(name, units, long_name, missing_value, minimum, maximum, &
                                      initial_value=initial_value, background_value=background_value, &
                                      specific_light_extinction=specific_light_extinction, &
                                      no_precipitation_dilution=no_precipitation_dilution, no_river_dilution=no_river
_dilution, &
                                      standard_variable=standard_variable, presence=presence, source=source_state, &
                                      state_index=id%state_index, read_index=id%index, &
                                      background=id%background, link=id%link)

      call register_source(self, id%link, id%sms)
      call register_surface_flux(self, id%link, id%surface_flux)
      call register_bottom_flux(self, id%link, id%bottom_flux)
      call register_movement(self, id%link, id%movement, vertical_movement)
   end subroutine register_interior_state_variable

   subroutine register_source(self, link, sms_id, source)
      class (type_base_model),           intent(inout)         :: self
      type (type_link),                  intent(inout)         :: link
      type (type_add_id),                intent(inout), target :: sms_id
      integer, optional,                 intent(in)            :: source

      integer                   :: source_
      type (type_link), pointer :: link2

      source_ = source_do
      if (present(source)) source_ = source
      if (.not. self%implements(source_)) source_ = source_constant
      if (.not. associated(sms_id%link)) call self%add_interior_variable(trim(link%name)//'_sms', &
         trim(link%target%units)//'/s', trim(link%target%long_name)//' sources-sinks', fill_value=0.0_rk, &
         missing_value=0.0_rk, output=output_none, write_index=sms_id%sum_index, source=source_, link=sms_id%link)
```

```
      sms_id%link%target%write_operator = operator_add
      link2 => link%target%sms_list%append(sms_id%link%target, sms_id%link%target%name)
      link%target%sms => link2
   end subroutine register_source

   subroutine register_surface_flux(self, link, surface_flux_id, source)
      class (type_base_model),       intent(inout)         :: self
      type (type_link),              intent(inout)         :: link
      type (type_horizontal_add_id), intent(inout), target :: surface_flux_id
      integer, optional,             intent(in)            :: source

      integer                   :: source_
      type (type_link), pointer :: link2

      source_ = source_do_surface
      if (present(source)) source_ = source
      if (.not. self%implements(source_)) source_ = source_constant
      if (.not. associated(surface_flux_id%link)) call self%add_horizontal_variable(trim(link%name) // '_sfl', &
         trim(link%target%units) // '*m/s', trim(link%target%long_name) // ' surface flux', fill_value=0.0_rk, &
         missing_value=0.0_rk, output=output_none, write_index=surface_flux_id%horizontal_sum_index, &
         domain=domain_surface, source=source_, link=surface_flux_id%link)
      surface_flux_id%link%target%write_operator = operator_add
      link2 => link%target%surface_flux_list%append(surface_flux_id%link%target, surface_flux_id%link%target%name)
      link%target%surface_flux => link2
   end subroutine register_surface_flux

   subroutine register_bottom_flux(self, link, bottom_flux_id, source)
      class (type_base_model),       intent(inout)         :: self
      type (type_link),              intent(inout)         :: link
      type (type_horizontal_add_id), intent(inout), target :: bottom_flux_id
      integer, optional,             intent(in)            :: source

      integer                   :: source_
      type (type_link), pointer :: link2

      source_ = source_do_bottom
      if (present(source)) source_ = source
      if (.not. self%implements(source_)) source_ = source_constant
      if (.not. associated(bottom_flux_id%link)) call self%add_horizontal_variable(trim(link%name) // '_bfl', &
         trim(link%target%units) // '*m/s', trim(link%target%long_name) // ' bottom flux', fill_value=0.0_rk, &
         missing_value=0.0_rk, output=output_none, write_index=bottom_flux_id%horizontal_sum_index, &
         domain=domain_bottom, source=source_, link=bottom_flux_id%link)
      bottom_flux_id%link%target%write_operator = operator_add
      link2 => link%target%bottom_flux_list%append(bottom_flux_id%link%target, bottom_flux_id%link%target%name)
      link%target%bottom_flux => link2
   end subroutine register_bottom_flux

   subroutine register_movement(self, link, movement_id, vertical_movement)
      class (type_base_model), intent(inout)         :: self
      type (type_link),        intent(inout)         :: link
      type (type_add_id),      intent(inout), target :: movement_id
      real(rk),                intent(in), optional  :: vertical_movement

      real(rk)                  :: vertical_movement_
      type (type_link), pointer :: link2

      vertical_movement_ = 0.0_rk
      if (present(vertical_movement)) vertical_movement_ = vertical_movement
      if (.not. associated(movement_id%link)) call self%add_interior_variable(trim(link%name) // '_w', &
         'm/s', trim(link%target%long_name) // ' vertical velocity', fill_value=vertical_movement_, missing_value=0.0
_rk, &
         output=output_none, write_index=movement_id%sum_index, link=movement_id%link, source=source_constant)
      if (self%implements(source_get_vertical_movement)) then
         movement_id%link%target%source = source_get_vertical_movement
         movement_id%link%target%write_operator = operator_add
      end if
      link2 => link%target%movement_list%append(movement_id%link%target, movement_id%link%target%name)
   end subroutine register_movement

   subroutine register_surface_source(self, link, sms_id, source)
      class (type_base_model),       intent(inout)         :: self
      type (type_link),              intent(inout)         :: link
      type (type_horizontal_add_id), intent(inout), target :: sms_id
      integer, optional,             intent(in)            :: source

      integer                   :: source_
      type (type_link), pointer :: link2

      source_ = source_do_surface
      if (present(source)) source_ = source
      if (.not. self%implements(source_)) source_ = source_constant
      if (.not. associated(sms_id%link)) call self%add_horizontal_variable(trim(link%name) // '_sms', &
         trim(link%target%units) // '/s', trim(link%target%long_name) // ' sources-sinks', fill_value=0.0_rk, &
         missing_value=0.0_rk, output=output_none, write_index=sms_id%horizontal_sum_index, link=sms_id%link, &
         domain=domain_surface, source=source_)
      sms_id%link%target%write_operator = operator_add
      link2 => link%target%sms_list%append(sms_id%link%target, sms_id%link%target%name)
      link%target%sms => link2
   end subroutine register_surface_source

   subroutine register_bottom_source(self, link, sms_id, source)
      class (type_base_model),       intent(inout)         :: self
      type (type_link),              intent(inout)         :: link
      type (type_horizontal_add_id), intent(inout), target :: sms_id
      integer, optional,             intent(in)            :: source

      integer                   :: source_
      type (type_link), pointer :: link2
```

```
1711
1712        source_ = source_do_bottom
1713        if (present(source)) source_ = source
1714        if (.not. self%implements(source_)) source_ = source_constant
1715        if (.not. associated(sms_id%link)) call self%add_horizontal_variable(trim(link%name) // '_sms', &
1716           trim(link%target%units) // '/s', trim(link%target%long_name) // ' sources-sinks', fill_value=0.0_rk, &
1717           missing_value=0.0_rk, output=output_none, write_index=sms_id%horizontal_sum_index, link=sms_id%link, &
1718           domain=domain_bottom, source=source_)
1719        sms_id%link%target%write_operator = operator_add
1720        link2 => link%target%sms_list%append(sms_id%link%target, sms_id%link%target%name)
1721        link%target%sms => link2
1722     end subroutine register_bottom_source
1723
1724     subroutine register_bottom_state_variable(self, id, name, units, long_name, &
1725                                               initial_value, minimum, maximum, missing_value, &
1726                                               standard_variable, presence, background_value)
1727        class (type_base_model),             intent(inout)         :: self
1728        type (type_bottom_state_variable_id), intent(inout), target :: id
1729        character(len=*),                    intent(in)            :: name, long_name, units
1730        real(rk),                            intent(in), optional  :: initial_value
1731        real(rk),                            intent(in), optional  :: minimum, maximum, missing_value, background_valu
e
1732        class (type_base_standard_variable),  intent(in), optional  :: standard_variable
1733        integer,                             intent(in), optional  :: presence
1734
1735        call self%add_horizontal_variable(name, units, long_name, missing_value, minimum, maximum, &
1736                                          initial_value=initial_value, background_value=background_value, &
1737                                          standard_variable=standard_variable, presence=presence, domain=domain_bottom,
 &
1738                                          state_index=id%bottom_state_index, read_index=id%horizontal_index, &
1739                                          background=id%background, link=id%link, source=source_state)
1740        call register_bottom_source(self, id%link, id%bottom_sms)
1741     end subroutine register_bottom_state_variable
1742
1743     subroutine register_surface_state_variable(self, id, name, units, long_name, &
1744                                                initial_value, minimum, maximum, missing_value, &
1745                                                standard_variable, presence, background_value)
1746        class (type_base_model),               intent(inout)         :: self
1747        type (type_surface_state_variable_id), intent(inout), target :: id
1748        character(len=*),                      intent(in)            :: name, long_name, units
1749        real(rk),                              intent(in), optional  :: initial_value
1750        real(rk),                              intent(in), optional  :: minimum, maximum, missing_value, background_val
ue
1751        class (type_base_standard_variable),   intent(in), optional  :: standard_variable
1752        integer,                               intent(in), optional  :: presence
1753
1754        call self%add_horizontal_variable(name, units, long_name, missing_value, minimum, maximum, &
1755                                          initial_value=initial_value, background_value=background_value, &
1756                                          standard_variable=standard_variable, presence=presence, domain=domain_surface
, &
1757                                          state_index=id%surface_state_index, read_index=id%horizontal_index, &
1758                                          background=id%background, link=id%link, source=source_state)
1759        call register_surface_source(self, id%link, id%surface_sms)
1760     end subroutine register_surface_state_variable
1761
1762     subroutine add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
1763                             initial_value, background_value, fill_value, standard_variable, presence, output, source,
&
1764                             act_as_state_variable, read_index, state_index, write_index, background, link)
1765        class (type_base_model),       target,intent(inout)       :: self
1766        type (type_internal_variable),pointer                     :: variable
1767        character(len=*),              target,intent(in)          :: name
1768        character(len=*),                     intent(in), optional :: long_name, units
1769        real(rk),                             intent(in), optional :: minimum, maximum, missing_value
1770        real(rk),                             intent(in), optional :: initial_value, background_value, fill_value
1771        class (type_base_standard_variable),  intent(in), optional :: standard_variable
1772        integer,                              intent(in), optional :: presence, output, source
1773        logical,                              intent(in), optional :: act_as_state_variable
1774        integer,                      target,            optional :: read_index, state_index, write_index
1775        real(rk),                     target,            optional :: background
1776        type (type_link),             pointer,           optional :: link
1777
1778        integer                   :: length, i
1779        character(len=256)        :: text
1780        type (type_link), pointer :: link_
1781        class (type_base_standard_variable), pointer :: pstandard_variable
1782
1783        ! Check whether the model information may be written to (only during initialization)
1784        if (self%frozen) call self%fatal_error('add_variable', &
1785           'Cannot register variable "' // trim(name) // '" because the model initialization phase has already complete
d &
1786           &(initialize has been called).')
1787
1788        ! Ascertain whether the provided name is valid.
1789        length = len_trim(name)
1790        if (length > len(variable%name)) then
1791           call self%fatal_error('add_variable', 'Variable name "' // trim(name) // '" exceeds maximum length.')
1792        elseif (length == 0) then
1793           call self%fatal_error('add_variable', 'Cannot register variable with empty name "".')
1794        elseif (name(length:length) == '*') then
1795           ! Last character is an asterisk (*) that needs to be replaced with an integer than makes the name unique.
1796           i = 1
1797           do
1798              write (variable%name,'(A,I0)') name(:length - 1), i
1799              if (.not. associated(self%links%find(variable%name))) exit
1800              i = i + 1
1801           end do
1802        elseif (name /= get_safe_name(name)) then
```

```
1803          call self%fatal_error('add_variable', 'Cannot register variable "' // trim(name) // '" because its name is n
      ot valid. &
1804               &Variable names can contain letters, digits and underscores only.')
1805       else
1806          variable%name = name
1807       end if
1808
1809       if (present(write_index) .and. .not. present(source)) call self%fatal_error('add_variable', &
1810          'Cannot register writable variable "' // trim(name) // '" because "source" argument is not provided.')
1811
1812       variable%owner => self
1813       if (present(units)) variable%units = units
1814       if (present(long_name)) then
1815          variable%long_name = long_name
1816       else
1817          variable%long_name = variable%name
1818       end if
1819       if (present(minimum))       variable%minimum       = minimum
1820       if (present(maximum))       variable%maximum       = maximum
1821       if (present(missing_value)) variable%missing_value = missing_value
1822       if (present(initial_value)) variable%initial_value = initial_value
1823       if (present(presence))      variable%presence      = presence
1824       if (present(act_as_state_variable)) variable%fake_state_variable = act_as_state_variable
1825       if (present(output))        variable%output        = output
1826       if (present(source))        variable%source        = source
1827       variable%prefill_value = variable%missing_value
1828       if (present(fill_value)) then
1829          variable%prefill = prefill_constant
1830          variable%prefill_value = fill_value
1831       end if
1832       if (present(standard_variable)) then
1833          pstandard_variable => standard_variable%resolve()
1834          select type (pstandard_variable)
1835          class is (type_domain_specific_standard_variable)
1836             call variable%standard_variables%add(pstandard_variable)
1837          class is (type_universal_standard_variable)
1838             select case (variable%domain)
1839             case (domain_interior);   call variable%standard_variables%add(pstandard_variable%in_interior())
1840             case (domain_surface);    call variable%standard_variables%add(pstandard_variable%at_surface())
1841             case (domain_bottom);     call variable%standard_variables%add(pstandard_variable%at_bottom())
1842             case (domain_horizontal); call variable%standard_variables%add(pstandard_variable%at_interfaces())
1843             end select
1844          end select
1845       end if
1846
1847       if (present(state_index)) then
1848          ! Ensure that initial value falls within prescribed valid range.
1849          if (variable%initial_value < variable%minimum .or. variable%initial_value > variable%maximum) then
1850             write (text,*) 'Initial value', variable%initial_value, 'for variable "' // trim(name) // '" lies&
1851                   &outside allowed range', variable%minimum, 'to', variable%maximum
1852             call self%fatal_error('fill_internal_variable', text)
1853          end if
1854
1855          ! Store a pointer to the variable that should hold the state variable index.
1856          call variable%state_indices%append(state_index)
1857       end if
1858
1859       if (present(background)) then
1860          ! Store a pointer to the variable that should hold the background value.
1861          ! If the background value itself is also prescribed, use it.
1862          call variable%background_values%append(background)
1863          if (present(background_value)) call variable%background_values%set_value(background_value)
1864       end if
1865
1866       if (present(read_index)) then
1867          variable%read_index => read_index
1868          call variable%read_indices%append(read_index)
1869       end if
1870       if (present(write_index)) then
1871          variable%write_index => write_index
1872          _ASSERT_(variable%source /= source_state, 'add_variable', 'Variable ' // trim(name) // ' being registered wi
      th source_state and write index.')
1873          call variable%write_indices%append(write_index)
1874       end if
1875
1876       ! Create a class pointer and use that to create a link.
1877       link_ => add_object(self, variable)
1878       if (present(link)) then
1879          if (associated(link)) call self%fatal_error('add_variable', 'Identifier supplied for ' // trim(name) // ' is
       already associated with ' // trim(link%name) // '.')
1880          link => link_
1881       end if
1882    end subroutine add_variable
1883
1884    subroutine add_interior_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &
1885                                           background_value, fill_value, specific_light_extinction, &
1886                                           no_precipitation_dilution, no_river_dilution, standard_variable, presence,
      output, &
1887                                           act_as_state_variable, source, &
1888                                           read_index, state_index, write_index, &
1889                                           background, link)
1890       class (type_base_model),target,      intent(inout)        :: self
1891       character(len=*),                    intent(in)           :: name
1892       character(len=*),                    intent(in), optional :: units, long_name
1893       real(rk),                            intent(in), optional :: minimum, maximum, missing_value
1894       real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
1895       real(rk),                            intent(in), optional :: specific_light_extinction
1896       logical,                             intent(in), optional :: no_precipitation_dilution, no_river_dilution
```

```
1897        class (type_base_standard_variable), intent(in), optional :: standard_variable
1898        integer,                             intent(in), optional :: presence, output, source
1899        logical,                             intent(in), optional :: act_as_state_variable
1900        integer,  target,                                optional :: read_index, state_index, write_index
1901        real(rk), target,                                optional :: background
1902        type (type_link), pointer,                       optional :: link
1903
1904        type (type_internal_variable), pointer :: variable
1905
1906        allocate(variable)
1907        variable%domain = domain_interior
1908
1909        ! Fill fields specific to interior variables.
1910        if (present(no_precipitation_dilution)) variable%no_precipitation_dilution = no_precipitation_dilution
1911        if (present(no_river_dilution))         variable%no_river_dilution         = no_river_dilution
1912        if (present(specific_light_extinction)) call variable%contributions%add( &
1913           standard_variables%attenuation_coefficient_of_photosynthetic_radiative_flux, scale_factor=specific_light_ext
    inction)
1914
1915        ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
1916        call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
1917                          initial_value, background_value, fill_value, standard_variable, presence, output, source, &
1918                          act_as_state_variable, read_index, state_index, write_index, background, link)
1919     end subroutine add_interior_variable
1920
1921     subroutine add_horizontal_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &

1922                                        background_value, fill_value, standard_variable, presence, output, &
1923                                        act_as_state_variable, domain, source, &
1924                                        read_index, state_index, write_index, background, link)
1925        class (type_base_model),target,      intent(inout)        :: self
1926        character(len=*),                    intent(in)           :: name
1927        character(len=*),                    intent(in), optional :: units, long_name
1928        real(rk),                            intent(in), optional :: minimum, maximum, missing_value
1929        real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
1930        class (type_base_standard_variable), intent(in), optional :: standard_variable
1931        integer,                             intent(in), optional :: presence, domain, output, source
1932        logical,                             intent(in), optional :: act_as_state_variable
1933        integer,  target,                                optional :: read_index, state_index, write_index
1934        real(rk), target,                                optional :: background
1935        type (type_link), pointer,                       optional :: link
1936
1937        type (type_internal_variable), pointer :: variable
1938
1939        allocate(variable)
1940        variable%domain = domain_horizontal
1941        if (present(domain)) variable%domain = domain
1942
1943        ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
1944        call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
1945                          initial_value, background_value, fill_value, standard_variable, presence, output, source, &
1946                          act_as_state_variable, read_index, state_index, write_index, background, link)
1947     end subroutine add_horizontal_variable
1948
1949     subroutine add_scalar_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &
1950                                    background_value, fill_value, standard_variable, presence, output, &
1951                                    read_index, state_index, write_index, sms_index, background, link)
1952        class (type_base_model),target,      intent(inout)        :: self
1953        character(len=*),                    intent(in)           :: name
1954        character(len=*),                    intent(in), optional :: units, long_name
1955        real(rk),                            intent(in), optional :: minimum, maximum, missing_value
1956        real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
1957        class (type_base_standard_variable), intent(in), optional :: standard_variable
1958        integer,                             intent(in), optional :: presence, output
1959        integer,  target,                                optional :: read_index, state_index, write_index, sms_index
1960        real(rk), target,                                optional :: background
1961        type (type_link), pointer,                       optional :: link
1962
1963        type (type_internal_variable), pointer :: variable
1964
1965        allocate(variable)
1966        variable%domain = domain_scalar
1967
1968        ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
1969        call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
1970                          initial_value, background_value, fill_value, standard_variable, presence, output, source_unkn
    own, &
1971                          .false., read_index, state_index, write_index, background, link)
1972     end subroutine add_scalar_variable
1973
1974     recursive function add_object(self, object) result(link)
1975        ! This subroutine creates a link to the supplied object, then allows
1976        ! parent models to do the same.
1977        ! NB this subroutine MUST be recursive, to allow parent models to override
1978        ! the properties of objects added by their child models.
1979        class (type_base_model), target, intent(inout) :: self
1980        type (type_internal_variable), pointer         :: object
1981
1982        type (type_link), pointer         :: link, parent_link
1983        character(len=attribute_length)   :: oriname
1984        integer                           :: instance
1985        logical                           :: duplicate
1986        type (type_link_pointer), pointer :: link_pointer
1987
1988        ! First check if a link with this name exists.
1989        duplicate = associated(self%links%find(object%name))
1990
1991        if (duplicate) then
```

```
1992          ! Link with this name exists already.
1993          ! Append numbers to the variable name until a unique name is found.
1994          oriname = object%name
1995          instance = 0
1996          do
1997             write (object%name,'(a,a,i0)') trim(oriname), '_', instance
1998             if (.not. associated(self%links%find(object%name))) exit
1999             instance = instance + 1
2000          end do
2001       end if
2002
2003       ! Create link for this object.
2004       link => self%links%append(object, object%name)
2005
2006       ! Store a pointer to the link with the object to facilitate redirection of the link during coupling.
2007       allocate(link_pointer)
2008       link_pointer%p => link
2009       link_pointer%next => object%first_link
2010       object%first_link => link_pointer
2011
2012       ! If this name matched that of a previous variable, create a coupling to it.
2013       if (duplicate) call self%request_coupling(link, oriname)
2014
2015       ! Forward to parent
2016       if (associated(self%parent)) then
2017          if (len_trim(self%name) + 1 + len_trim(object%name) > len(object%name)) call self%fatal_error('add_object',
   &
2018             'Variable path "' // trim(self%name) // '/' // trim(object%name) // '" exceeds maximum allowed length.')
2019          object%name = trim(self%name) // '/' // trim(object%name)
2020
2021          ! Below, the equivalent self%parent%add_object(object) confuses PGI 18.10 (Jorn 2019-04-24)
2022          parent_link => add_object(self%parent, object)
2023       end if
2024    end function add_object
2025
2026    subroutine register_interior_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variabl
   e, output, &
2027                                                     source, act_as_state_variable, prefill_value)
2028       class (type_base_model),             intent(inout), target :: self
2029       type (type_diagnostic_variable_id),  intent(inout), target :: id
2030       character(len=*),                    intent(in)           :: name, long_name, units
2031       integer,                             intent(in), optional :: output, source
2032       real(rk),                            intent(in), optional :: missing_value, prefill_value
2033       class (type_base_standard_variable), intent(in), optional :: standard_variable
2034       logical,                             intent(in), optional :: act_as_state_variable
2035
2036       integer :: source_
2037
2038       source_ = source_do
2039       if (present(source)) source_ = source
2040       call self%add_interior_variable(name, units, long_name, missing_value, fill_value=prefill_value, &
2041          standard_variable=standard_variable, output=output, source=source_, write_index=id%write_index, link=id%link
   , &
2042          act_as_state_variable=act_as_state_variable)
2043    end subroutine register_interior_diagnostic_variable
2044
2045    subroutine register_horizontal_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_varia
   ble, output, &
2046                                                       source, act_as_state_variable, domain)
2047       class (type_base_model),                      intent(inout), target :: self
2048       type (type_horizontal_diagnostic_variable_id), intent(inout), target :: id
2049       character(len=*),                             intent(in)            :: name, units, long_name
2050       integer,                                      intent(in)            :: source
2051       integer,                                      intent(in), optional  :: output, domain
2052       real(rk),                                     intent(in), optional  :: missing_value
2053       class (type_base_standard_variable),          intent(in), optional  :: standard_variable
2054       logical,                                      intent(in), optional  :: act_as_state_variable
2055
2056       call self%add_horizontal_variable(name, units, long_name, missing_value, &
2057                                         standard_variable=standard_variable, output=output, &
2058                                         source=source, write_index=id%horizontal_write_index, link=id%link, &
2059                                         act_as_state_variable=act_as_state_variable, domain=domain)
2060    end subroutine register_horizontal_diagnostic_variable
2061
2062    subroutine register_surface_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variable
   , &
2063                                                    output, source, act_as_state_variable)
2064       class (type_base_model),                    intent(inout), target :: self
2065       type (type_surface_diagnostic_variable_id), intent(inout), target :: id
2066       character(len=*),                           intent(in)            :: name, units, long_name
2067       integer,                                    intent(in), optional  :: output, source
2068       real(rk),                                   intent(in), optional  :: missing_value
2069       class (type_base_standard_variable),        intent(in), optional  :: standard_variable
2070       logical,                                    intent(in), optional  :: act_as_state_variable
2071
2072       integer :: source_
2073
2074       source_ = source_do_surface
2075       if (present(source)) source_ = source
2076       call self%add_horizontal_variable(name, units, long_name, missing_value, &
2077                                         standard_variable=standard_variable, output=output, &
2078                                         source=source_, write_index=id%surface_write_index, link=id%link, &
2079                                         act_as_state_variable=act_as_state_variable, domain=domain_surface)
2080    end subroutine register_surface_diagnostic_variable
2081
2082    subroutine register_bottom_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variable,
    &
2083                                                   output, source, act_as_state_variable)
```

```
      class (type_base_model),                   intent(inout), target :: self
      type (type_bottom_diagnostic_variable_id), intent(inout), target :: id
      character(len=*),                          intent(in)            :: name, units, long_name
      integer,                                   intent(in), optional  :: output, source
      real(rk),                                  intent(in), optional  :: missing_value
      class (type_base_standard_variable),       intent(in), optional  :: standard_variable
      logical,                                   intent(in), optional  :: act_as_state_variable

      integer :: source_

      source_ = source_do_bottom
      if (present(source)) source_ = source
      call self%add_horizontal_variable(name, units, long_name, missing_value, &
                                        standard_variable=standard_variable, output=output, &
                                        source=source_, write_index=id%bottom_write_index, link=id%link, &
                                        act_as_state_variable=act_as_state_variable, domain=domain_bottom)
   end subroutine register_bottom_diagnostic_variable

   subroutine register_interior_state_dependency(self, id, name, units, long_name, required)
      class (type_base_model),               intent(inout)         :: self
      type (type_state_variable_id),         intent(inout), target :: id
      character(len=*),                      intent(in)            :: name, units, long_name
      logical,                               intent(in), optional  :: required

      integer :: presence

      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if
      call register_interior_state_variable(self, id, name, units, long_name, presence=presence)
   end subroutine register_interior_state_dependency

   subroutine register_bottom_state_dependency(model, id, name, units, long_name, required)
      class (type_base_model),                 intent(inout)         :: model
      type (type_bottom_state_variable_id),    intent(inout), target :: id
      character(len=*),                        intent(in)            :: name, units, long_name
      logical,                                 intent(in), optional  :: required

      integer :: presence

      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if
      call register_bottom_state_variable(model, id, name, units, long_name, presence=presence)
   end subroutine register_bottom_state_dependency

   subroutine register_surface_state_dependency(model, id, name, units, long_name, required)
      class (type_base_model),                 intent(inout)         :: model
      type (type_surface_state_variable_id),   intent(inout), target :: id
      character(len=*),                        intent(in)            :: name, units, long_name
      logical,                                 intent(in), optional  :: required

      integer :: presence

      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if
      call register_surface_state_variable(model, id, name, units, long_name, presence=presence)
   end subroutine register_surface_state_dependency

   subroutine register_standard_interior_state_dependency(self, id, standard_variable, required)
      class (type_base_model),                intent(inout) :: self
      type (type_state_variable_id), target,  intent(inout) :: id
      type (type_interior_standard_variable), intent(in)    :: standard_variable
      logical, optional,                      intent(in)    :: required

      call register_interior_state_dependency(self, id, standard_variable%name, standard_variable%units, standard_var
iable%name, &
         required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_interior_state_dependency

   subroutine register_standard_bottom_state_dependency(self, id, standard_variable, required)
      class (type_base_model),                      intent(inout) :: self
      type (type_bottom_state_variable_id), target, intent(inout) :: id
      type (type_bottom_standard_variable),         intent(in)    :: standard_variable
      logical, optional,                            intent(in)    :: required

      call register_bottom_state_dependency(self, id, standard_variable%name, standard_variable%units, standard_varia
ble%name, &
         required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_bottom_state_dependency

   subroutine register_standard_bottom_state_dependency2(self, id, standard_variable, required)
      class (type_base_model),                      intent(inout) :: self
      type (type_bottom_state_variable_id), target, intent(inout) :: id
      type (type_horizontal_standard_variable),     intent(in)    :: standard_variable
      logical, optional,                            intent(in)    :: required

      call register_bottom_state_dependency(self, id, standard_variable%name, standard_variable%units, standard_varia
ble%name, &
         required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_bottom_state_dependency2
```

```
2179
2180    subroutine register_standard_surface_state_dependency(self, id, standard_variable, required)
2181       class (type_base_model),                      intent(inout) :: self
2182       type (type_surface_state_variable_id), target, intent(inout) :: id
2183       type (type_surface_standard_variable),         intent(in)    :: standard_variable
2184       logical, optional,                             intent(in)    :: required
2185
2186       call register_surface_state_dependency(self, id, standard_variable%name, standard_variable%units, standard_vari
able%name, &
2187          required=required)
2188       call self%request_coupling(id, standard_variable)
2189    end subroutine register_standard_surface_state_dependency
2190
2191    subroutine register_standard_surface_state_dependency2(self, id, standard_variable, required)
2192       class (type_base_model),                      intent(inout) :: self
2193       type (type_surface_state_variable_id), target, intent(inout) :: id
2194       type (type_horizontal_standard_variable),      intent(in)    :: standard_variable
2195       logical, optional,                             intent(in)    :: required
2196
2197       call register_surface_state_dependency(self, id, standard_variable%name, standard_variable%units, standard_vari
able%name, &
2198          required=required)
2199       call self%request_coupling(id, standard_variable)
2200    end subroutine register_standard_surface_state_dependency2
2201
2202    subroutine register_standard_interior_dependency(self, id, standard_variable, required)
2203       class (type_base_model),                intent(inout) :: self
2204       type (type_dependency_id), target,      intent(inout) :: id
2205       type (type_interior_standard_variable), intent(in)    :: standard_variable
2206       logical, optional,                      intent(in)    :: required
2207
2208       call register_named_interior_dependency(self, id, standard_variable%name, standard_variable%units, standard_var
iable%name, &
2209                                               required=required)
2210       call self%request_coupling(id, standard_variable)
2211    end subroutine register_standard_interior_dependency
2212
2213    subroutine register_universal_interior_dependency(self, id, standard_variable, required)
2214       class (type_base_model),                 intent(inout) :: self
2215       type (type_dependency_id), target,       intent(inout) :: id
2216       type (type_universal_standard_variable), intent(in)    :: standard_variable
2217       logical, optional,                       intent(in)    :: required
2218
2219       call register_standard_interior_dependency(self, id, standard_variable%in_interior(), required)
2220    end subroutine register_universal_interior_dependency
2221
2222    subroutine register_standard_horizontal_dependency(self, id, standard_variable, required)
2223       class (type_base_model),                  intent(inout)         :: self
2224       type (type_horizontal_dependency_id),     intent(inout), target :: id
2225       type (type_horizontal_standard_variable), intent(in)            :: standard_variable
2226       logical, optional,                        intent(in)            :: required
2227
2228       call register_named_horizontal_dependency(self, id, standard_variable%name, standard_variable%units, standard_v
ariable%name, &
2229                                                 required=required)
2230       call self%request_coupling(id, standard_variable)
2231    end subroutine register_standard_horizontal_dependency
2232
2233    subroutine register_standard_horizontal_dependency2(self, id, standard_variable, required)
2234       class (type_base_model),                intent(inout)         :: self
2235       type (type_horizontal_dependency_id),  intent(inout), target :: id
2236       type (type_surface_standard_variable), intent(in)            :: standard_variable
2237       logical, optional,                      intent(in)            :: required
2238
2239       call register_named_horizontal_dependency(self, id, standard_variable%name, standard_variable%units, standard_v
ariable%name, &
2240                                                 required=required)
2241       call self%request_coupling(id, standard_variable)
2242    end subroutine register_standard_horizontal_dependency2
2243
2244    subroutine register_standard_horizontal_dependency3(self, id, standard_variable, required)
2245       class (type_base_model),                intent(inout)         :: self
2246       type (type_horizontal_dependency_id), intent(inout), target :: id
2247       type (type_bottom_standard_variable), intent(in)            :: standard_variable
2248       logical, optional,                      intent(in)            :: required
2249
2250       call register_named_horizontal_dependency(self, id, standard_variable%name, standard_variable%units, standard_v
ariable%name, &
2251                                                 required=required)
2252       call self%request_coupling(id, standard_variable)
2253    end subroutine register_standard_horizontal_dependency3
2254
2255    subroutine register_universal_horizontal_dependency(self, id, standard_variable, domain, required)
2256       class (type_base_model),                 intent(inout)         :: self
2257       type (type_horizontal_dependency_id),    intent(inout), target :: id
2258       type (type_universal_standard_variable), intent(in)            :: standard_variable
2259       integer, optional,                       intent(in)            :: domain
2260       logical, optional,                       intent(in)            :: required
2261
2262       integer :: domain_
2263
2264       domain_ = domain_horizontal
2265       if (present(domain)) domain_ = domain
2266       select case (domain_)
2267       case (domain_surface);    call register_standard_horizontal_dependency2(self, id, standard_variable%at_surface(
), required)
2268       case (domain_bottom);     call register_standard_horizontal_dependency3(self, id, standard_variable%at_bottom()
, required)
```

```
      case (domain_horizontal); call register_standard_horizontal_dependency(self, id, standard_variable%at_interfaces(), required)
      case default
         call self%fatal_error('register_universal_horizontal_dependency', 'Specified domain must be domain_surface, domain_bottom, or domain_horizontal.')
      end select
   end subroutine register_universal_horizontal_dependency

   subroutine register_standard_surface_dependency(self, id, standard_variable, required)
      class (type_base_model),               intent(inout)         :: self
      type (type_surface_dependency_id),     intent(inout), target :: id
      type (type_surface_standard_variable), intent(in)            :: standard_variable
      logical, optional,                     intent(in)            :: required

      call register_named_surface_dependency(self, id, standard_variable%name, standard_variable%units, standard_variable%name, &
                                             required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_surface_dependency

   subroutine register_standard_surface_dependency2(self, id, standard_variable, required)
      class (type_base_model),                 intent(inout)         :: self
      type (type_surface_dependency_id),       intent(inout), target :: id
      type (type_horizontal_standard_variable), intent(in)           :: standard_variable
      logical, optional,                       intent(in)            :: required

      call register_named_surface_dependency(self, id, standard_variable%name, standard_variable%units, standard_variable%name, &
                                             required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_surface_dependency2

   subroutine register_universal_surface_dependency(self, id, standard_variable, required)
      class (type_base_model),                 intent(inout)         :: self
      type (type_surface_dependency_id),       intent(inout), target :: id
      type (type_universal_standard_variable), intent(in)            :: standard_variable
      logical, optional,                       intent(in)            :: required

      call register_standard_surface_dependency(self, id, standard_variable%at_surface(), required)
   end subroutine register_universal_surface_dependency

   subroutine register_standard_bottom_dependency(self, id, standard_variable, required)
      class (type_base_model),               intent(inout)         :: self
      type (type_bottom_dependency_id),      intent(inout), target :: id
      type (type_bottom_standard_variable), intent(in)             :: standard_variable
      logical, optional,                     intent(in)            :: required

      call register_named_bottom_dependency(self, id, standard_variable%name, standard_variable%units, standard_variable%name, &
                                            required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_bottom_dependency

   subroutine register_standard_bottom_dependency2(self, id, standard_variable, required)
      class (type_base_model),                 intent(inout)         :: self
      type (type_bottom_dependency_id),        intent(inout), target :: id
      type (type_horizontal_standard_variable), intent(in)           :: standard_variable
      logical, optional,                       intent(in)            :: required

      call register_named_bottom_dependency(self, id, standard_variable%name, standard_variable%units, standard_variable%name, &
                                            required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_bottom_dependency2

   subroutine register_universal_bottom_dependency(self, id, standard_variable, required)
      class (type_base_model),                 intent(inout)         :: self
      type (type_bottom_dependency_id),        intent(inout), target :: id
      type (type_universal_standard_variable), intent(in)            :: standard_variable
      logical, optional,                       intent(in)            :: required

      call register_standard_bottom_dependency(self, id, standard_variable%at_bottom(), required)
   end subroutine register_universal_bottom_dependency

   subroutine register_standard_global_dependency(self, id, standard_variable, required)
      class (type_base_model),               intent(inout)         :: self
      type (type_global_dependency_id),      intent(inout), target :: id
      type (type_global_standard_variable), intent(in)             :: standard_variable
      logical, optional,                     intent(in)            :: required

      call register_named_global_dependency(self, id, standard_variable%name, standard_variable%units, standard_variable%name, &
                                            required=required)
      call self%request_coupling(id, standard_variable)
   end subroutine register_standard_global_dependency

   subroutine register_named_interior_dependency(self, id, name, units, long_name, required)
      class (type_base_model),                intent(inout)         :: self
      type (type_dependency_id),              intent(inout), target :: id
      character(len=*),                       intent(in)            :: name, units, long_name
      logical,                                intent(in), optional  :: required

      integer :: presence

      ! Dependencies MUST be fulfilled, unless explicitly specified that this is not so (required=.false.)
      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
```

```
      end if

      call self%add_interior_variable(name, units, long_name, presence=presence, &
         read_index=id%index, background=id%background, link=id%link)
   end subroutine register_named_interior_dependency

   subroutine register_named_horizontal_dependency(self, id, name, units, long_name, required)
      class (type_base_model),              intent(inout)        :: self
      type (type_horizontal_dependency_id), intent(inout), target :: id
      character(len=*),                     intent(in)           :: name, units, long_name
      logical,                              intent(in), optional  :: required

      integer :: presence

      ! Dependencies MUST be fulfilled, unless explicitly specified that this is not so (required=.false.)
      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if

      call self%add_horizontal_variable(name, units, long_name, presence=presence, &
         read_index=id%horizontal_index, background=id%background, link=id%link)
   end subroutine register_named_horizontal_dependency

   subroutine register_named_surface_dependency(self, id, name, units, long_name, required)
      class (type_base_model),           intent(inout)        :: self
      type (type_surface_dependency_id), intent(inout), target :: id
      character(len=*),                  intent(in)           :: name, units, long_name
      logical,                           intent(in), optional  :: required

      integer :: presence

      ! Dependencies MUST be fulfilled, unless explicitly specified that this is not so (required=.false.)
      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if

      call self%add_horizontal_variable(name, units, long_name, presence=presence, &
         read_index=id%horizontal_index, background=id%background, link=id%link, domain=domain_surface)
   end subroutine register_named_surface_dependency

   subroutine register_named_bottom_dependency(self, id, name, units, long_name, required)
      class (type_base_model),          intent(inout)         :: self
      type (type_bottom_dependency_id), intent(inout), target :: id
      character(len=*),                 intent(in)            :: name, units, long_name
      logical,                          intent(in), optional  :: required

      integer :: presence

      ! Dependencies MUST be fulfilled, unless explicitly specified that this is not so (required=.false.)
      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if

      call self%add_horizontal_variable(name, units, long_name, presence=presence, &
         read_index=id%horizontal_index, background=id%background, link=id%link, domain=domain_bottom)
   end subroutine register_named_bottom_dependency

   subroutine register_named_global_dependency(self, id, name, units, long_name, required)
      class (type_base_model),          intent(inout)         :: self
      type (type_global_dependency_id), intent(inout), target :: id
      character(len=*),                 intent(in)            :: name, units, long_name
      logical,                          intent(in), optional  :: required

      integer :: presence

      ! Dependencies MUST be fulfilled, unless explicitly specified that this is not so (required=.false.)
      presence = presence_external_required
      if (present(required)) then
         if (.not. required) presence = presence_external_optional
      end if

      call self%add_scalar_variable(name, units, long_name, presence=presence, &
         read_index=id%global_index, background=id%background, link=id%link)
   end subroutine register_named_global_dependency

   subroutine register_interior_expression_dependency(self, id, expression)
      class (type_base_model),           intent(inout) :: self
      type (type_dependency_id), target, intent(inout) :: id
      class (type_interior_expression),  intent(in)    :: expression

      class (type_interior_expression), allocatable :: copy

      allocate(copy, source=expression)
      copy%out => id%index
      call self%register_dependency(id, copy%output_name, '', copy%output_name)
      copy%output_name = id%link%target%name

      call register_expression(self,copy)
      deallocate(copy)
   end subroutine

   subroutine register_horizontal_expression_dependency(self, id, expression)
      class (type_base_model),              intent(inout)        :: self
      type (type_horizontal_dependency_id), intent(inout), target :: id
      class (type_horizontal_expression),   intent(in)           :: expression
```

```
2458
2459        class (type_horizontal_expression), allocatable :: copy
2460
2461        allocate(copy, source=expression)
2462        copy%out => id%horizontal_index
2463        call self%register_dependency(id, copy%output_name, '', copy%output_name)
2464        copy%output_name = id%link%target%name
2465
2466        call register_expression(self, copy)
2467        deallocate(copy)
2468     end subroutine
2469
2470     recursive subroutine register_expression(self, expression)
2471        class (type_base_model), intent(inout) :: self
2472        class (type_expression), intent(in)    :: expression
2473
2474        class (type_expression), pointer :: current
2475
2476        if (.not. associated(self%first_expression)) then
2477           allocate(self%first_expression, source=expression)
2478           current => self%first_expression
2479        else
2480           current => self%first_expression
2481           do while (associated(current%next))
2482              current => current%next
2483           end do
2484           allocate(current%next, source=expression)
2485           current => current%next
2486        end if
2487
2488        if (associated(self%parent)) call register_expression(self%parent, expression)
2489     end subroutine
2490
2491     subroutine get_real_parameter(self, value, name, units, long_name, default, scale_factor, minimum, maximum)
2492        class (type_base_model), intent(inout), target  :: self
2493        real(rk),                intent(inout)          :: value
2494        character(len=*),        intent(in)             :: name
2495        character(len=*),        intent(in),   optional :: units, long_name
2496        real(rk),                intent(in),   optional :: default, scale_factor, minimum, maximum
2497
2498        class (type_property), pointer :: property
2499        logical                        :: success
2500        type (type_real_property)      :: current_parameter
2501        character(len=13)              :: text1, text2
2502
2503        if (present(default)) then
2504           current_parameter%has_default = .true.
2505           current_parameter%default = default
2506           value = default
2507        end if
2508
2509        ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
2510        property => self%parameters%find_in_tree(name)
2511        if (associated(property)) then
2512           ! Value found - try to convert to real.
2513           value = property%to_real(success=success)
2514           if (.not. success) call self%fatal_error('get_real_parameter', &
2515              'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not a real number.'
)
2516        elseif (.not.present(default)) then
2517           call self%fatal_error('get_real_parameter', 'No value provided for parameter "' // trim(name) // '".')
2518        end if
2519
2520        if (present(minimum)) then
2521           if (value < minimum) then
2522              write (text1,'(G13.6)') value
2523              write (text2,'(G13.6)') minimum
2524              call self%fatal_error('get_real_parameter', 'Value ' // trim(adjustl(text1)) // ' for parameter "' // tri
m(name) &
2525                 // '" is less than prescribed minimum of ' // trim(adjustl(text2)) // '.')
2526           end if
2527        end if
2528        if (present(maximum)) then
2529           if (value > maximum) then
2530              write (text1,'(G13.6)') value
2531              write (text2,'(G13.6)') maximum
2532              call self%fatal_error('get_real_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // trim
(name) &
2533                 // '" exceeds prescribed maximum of ' // trim(adjustl(text2)) // '.')
2534           end if
2535        end if
2536
2537        ! Store parameter settings
2538        current_parameter%value = value
2539        call set_parameter(self, current_parameter, name, units, long_name)
2540
2541        ! Apply scale factor to value provided to the model (if requested).
2542        if (present(scale_factor)) value = value * scale_factor
2543     end subroutine get_real_parameter
2544
2545     subroutine set_parameter(self, parameter, name, units, long_name)
2546        class (type_base_model), intent(inout), target :: self
2547        class (type_property),   intent(inout)         :: parameter
2548        character(len=*),        intent(in)            :: name
2549        character(len=*),        intent(in), optional  :: units, long_name
2550
2551        parameter%name = name
2552        if (present(units))     parameter%units     = units
```

```
2553       if (present(long_name)) parameter%long_name = long_name
2554       call self%parameters%set_in_tree(parameter)
2555    end subroutine set_parameter
2556
2557    subroutine get_integer_parameter(self, value, name, units, long_name, default, minimum, maximum)
2558       class (type_base_model), intent(inout), target :: self
2559       integer,                 intent(inout)         :: value
2560       character(len=*),        intent(in)            :: name
2561       character(len=*),        intent(in), optional  :: units, long_name
2562       integer,                 intent(in), optional  :: default, minimum, maximum
2563
2564       class (type_property), pointer :: property
2565       type (type_integer_property)   :: current_parameter
2566       logical                        :: success
2567       character(len=8)               :: text1, text2
2568
2569       if (present(default)) then
2570          current_parameter%has_default = .true.
2571          current_parameter%default = default
2572          value = default
2573       end if
2574
2575       ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
2576       property => self%parameters%find_in_tree(name)
2577       if (associated(property)) then
2578          ! Value found - try to convert to integer.
2579          value = property%to_integer(success=success)
2580          if (.not. success) call self%fatal_error('get_integer_parameter', &
2581             'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not an integer numb
er.')
2582       elseif (.not.present(default)) then
2583          call self%fatal_error('get_integer_parameter', 'No value provided for parameter "' // trim(name) // '".')
2584       end if
2585
2586       if (present(minimum)) then
2587          if (value < minimum) then
2588             write (text1,'(I0)') value
2589             write (text2,'(I0)') minimum
2590             call self%fatal_error('get_integer_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // t
rim(name) &
2591                // '" is less than prescribed minimum of ' // trim(adjustl(text2)) // '.')
2592          end if
2593       end if
2594       if (present(maximum)) then
2595          if (value > maximum) then
2596             write (text1,'(I0)') value
2597             write (text2,'(I0)') maximum
2598             call self%fatal_error('get_integer_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // t
rim(name) &
2599                //'" exceeds prescribed maximum of ' // trim(adjustl(text2)) // '.')
2600          end if
2601       end if
2602
2603       ! Store parameter settings
2604       current_parameter%value = value
2605       call set_parameter(self, current_parameter, name, units, long_name)
2606    end subroutine get_integer_parameter
2607
2608    subroutine get_logical_parameter(self, value, name, units, long_name, default)
2609       class (type_base_model), intent(inout), target :: self
2610       logical,                 intent(inout)         :: value
2611       character(len=*),        intent(in)            :: name
2612       character(len=*),        intent(in), optional  :: units, long_name
2613       logical,                 intent(in), optional  :: default
2614
2615       class (type_property), pointer :: property
2616       type (type_logical_property)   :: current_parameter
2617       logical                        :: success
2618
2619       if (present(default)) then
2620          current_parameter%has_default = .true.
2621          current_parameter%default = default
2622          value = default
2623       end if
2624
2625       ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
2626       property => self%parameters%find_in_tree(name)
2627       if (associated(property)) then
2628          ! Value found - try to convert to logical.
2629          value = property%to_logical(success=success)
2630          if (.not. success) call self%fatal_error('get_logical_parameter', &
2631             'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not a Boolean value
.')
2632       elseif (.not. present(default)) then
2633          call self%fatal_error('get_logical_parameter', 'No value provided for parameter "' // trim(name) // '".')
2634       end if
2635
2636       ! Store parameter settings
2637       current_parameter%value = value
2638       call set_parameter(self, current_parameter, name, units, long_name)
2639    end subroutine get_logical_parameter
2640
2641    recursive subroutine get_string_parameter(self, value, name, units, long_name, default)
2642       class (type_base_model), intent(inout), target :: self
2643       character(len=*),        intent(inout)         :: value
2644       character(len=*),        intent(in)            :: name
2645       character(len=*),        intent(in), optional  :: units, long_name
2646       character(len=*),        intent(in), optional  :: default
```

```
2647
2648        class (type_property), pointer :: property
2649        type (type_string_property)    :: current_parameter
2650        logical                        :: success
2651
2652        if (present(default)) then
2653           current_parameter%has_default = .true.
2654           current_parameter%default = default
2655           value = default
2656        end if
2657
2658        ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
2659        property => self%parameters%find_in_tree(name)
2660        if (associated(property)) then
2661           ! Value found - try to convert to string.
2662           value = property%to_string(success=success)
2663           if (.not. success) call self%fatal_error('get_string_parameter', &
2664              'Value for parameter "' // trim(name) // '" cannot be converted to string.')
2665        elseif (.not. present(default)) then
2666           call self%fatal_error('get_string_parameter','No value provided for parameter "' // trim(name) // '".')
2667        end if
2668
2669        ! Store parameter settings
2670        current_parameter%value = value
2671        call set_parameter(self, current_parameter, name, units, long_name)
2672     end subroutine get_string_parameter
2673
2674     function find_object(self, name, recursive, exact) result(object)
2675        class (type_base_model),  intent(in), target :: self
2676        character(len=*),         intent(in)         :: name
2677        logical,        optional, intent(in)         :: recursive, exact
2678        type (type_internal_variable), pointer       :: object
2679
2680        type (type_link), pointer :: link
2681
2682        object => null()
2683        link => self%find_link(name, recursive, exact)
2684        if (associated(link)) object => link%target
2685
2686     end function find_object
2687
2688     recursive function find_link(self, name, recursive, exact) result(link)
2689        class (type_base_model),  intent(in), target :: self
2690        character(len=*),         intent(in)         :: name
2691        logical,        optional, intent(in)         :: recursive, exact
2692        type (type_link), pointer                    :: link
2693
2694        integer                          :: n
2695        logical                          :: recursive_eff, exact_eff
2696        class (type_base_model),pointer :: current
2697
2698        link => null()
2699
2700        n = len_trim(name)
2701        if (n >= 1) then
2702           if (name(1:1) == '/') then
2703              link => find_link(self, name(2:), recursive, exact=.true.)
2704              return
2705           end if
2706           if (n >= 2) then
2707              if (name(1:2) == './') then
2708                 link => find_link(self, name(3:), recursive, exact=.true.)
2709                 return
2710              end if
2711              if (n >= 3) then
2712                 if (name(1:3) == '../') then
2713                    if (.not. associated(self%parent)) return
2714                    link => find_link(self%parent, name(4:), recursive, exact=.true.)
2715                    return
2716                 end if
2717              end if
2718           end if
2719        end if
2720
2721        recursive_eff = .false.
2722        if (present(recursive)) recursive_eff = recursive
2723
2724        ! First search self and ancestors (if allowed) based on exact name provided.
2725        current => self
2726        do while (associated(current))
2727           link => current%links%find(name)
2728           if (associated(link)) return
2729           if (.not. recursive_eff) exit
2730           current => current%parent
2731        end do
2732
2733        exact_eff = .true.
2734        if (present(exact)) exact_eff = exact
2735        if (exact_eff) return
2736
2737        ! Not found. Now search self and ancestors (if allowed) based on safe name (letters and underscores only).
2738        current => self
2739        do while (associated(current))
2740           link => current%links%first
2741           do while (associated(link))
2742              if (get_safe_name(link%name) == name) return
2743              link => link%next
2744           end do
```

```
2745          if (.not. recursive_eff) exit
2746          current => current%parent
2747       end do
2748    end function find_link
2749
2750    function find_model(self, name, recursive) result(found_model)
2751       class (type_base_model), target, intent(in) :: self
2752       character(len=*),                intent(in) :: name
2753       logical, optional,               intent(in) :: recursive
2754       class (type_base_model), pointer            :: found_model
2755
2756       class (type_base_model), pointer     :: current_root
2757       logical                              :: recursive_eff
2758       type (type_model_list_node), pointer :: node
2759       integer                              :: istart, length
2760
2761       found_model => null()
2762
2763       ! Determine whether to also try among ancestors
2764       recursive_eff = .false.
2765       if (present(recursive)) recursive_eff = recursive .and. .not. (name == '.' .or. name == '..' &
2766             .or. name(:min(2, len(name))) == './' .or. name(:min(3, len(name))) == '../')
2767
2768       current_root => self
2769       do while (associated(current_root))
2770          ! Process individual path components (separated by /)
2771          found_model => current_root
2772          istart = 1
2773          do while (associated(found_model) .and. istart <= len(name))
2774             length = index(name(istart:), '/') - 1
2775             if (length == -1) length = len(name) - istart + 1
2776             if (length == 2 .and. name(istart:istart + length - 1) == '..') then
2777                found_model => found_model%parent
2778             elseif (.not. (length == 1 .and. name(istart:istart + length - 1) == '.')) then
2779                node => found_model%children%find(name(istart:istart + length - 1))
2780                found_model => null()
2781                if (associated(node)) found_model => node%model
2782             end if
2783             istart = istart + length + 1
2784          end do
2785
2786          ! Only continue if we have not found the model and are allowed to try parent model.
2787          if (associated(found_model) .or. .not. recursive_eff) return
2788
2789          current_root => current_root%parent
2790       end do
2791    end function find_model
2792
2793    function get_aggregate_variable_access(self, standard_variable) result(aggregate_variable_access)
2794       class (type_base_model),                        intent(inout) :: self
2795       class (type_domain_specific_standard_variable), target        :: standard_variable
2796
2797       type (type_aggregate_variable_access), pointer :: aggregate_variable_access
2798       logical,                               pointer :: pmember
2799
2800       ! First try to locate existing requests object for the specified standard variable.
2801       aggregate_variable_access => self%first_aggregate_variable_access
2802       pmember => standard_variable%aggregate_variable
2803       do while (associated(aggregate_variable_access))
2804          ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
     erences.
2805          if (associated(pmember, aggregate_variable_access%standard_variable%aggregate_variable)) return
2806          aggregate_variable_access => aggregate_variable_access%next
2807       end do
2808
2809       ! Not found - create a new requests object.
2810       allocate(aggregate_variable_access)
2811       aggregate_variable_access%standard_variable => standard_variable
2812       aggregate_variable_access%next => self%first_aggregate_variable_access
2813       self%first_aggregate_variable_access => aggregate_variable_access
2814    end function get_aggregate_variable_access
2815
2816    function get_free_unit() result(unit)
2817       integer :: unit
2818       integer, parameter :: LUN_MIN=10, LUN_MAX=1000
2819
2820       logical :: opened
2821
2822       do unit = LUN_MIN, LUN_MAX
2823          inquire(unit=unit, opened=opened)
2824          if (.not. opened) return
2825       end do
2826       unit = -1
2827    end function get_free_unit
2828
2829    function get_safe_name(name) result(safe_name)
2830       character(len=*), intent(in) :: name
2831       character(len=len(name))     :: safe_name
2832
2833       integer :: i, ch
2834       logical :: valid
2835
2836       safe_name = name
2837       do i = 1, len_trim(name)
2838          ch = iachar(name(i:i))
2839          valid = (ch >= iachar('a') .and. ch <= iachar('z')) & ! Lower-case letter
2840             .or. (ch >= iachar('A') .and. ch <= iachar('Z')) & ! Upper-case letter
2841             .or. (ch >= iachar('0') .and. ch <= iachar('9')) & ! Number
```

```
2842            .or. (ch == iachar('_'))                          ! Underscore
2843         if (.not. valid) safe_name(i:i) = '_'
2844      end do
2845   end function
2846
2847   recursive subroutine abstract_model_factory_initialize(self)
2848      class (type_base_model_factory), intent(inout) :: self
2849
2850      type (type_base_model_factory_node), pointer :: current
2851
2852      self%initialized = .true.
2853      current => self%first_child
2854      do while(associated(current))
2855         if (.not. current%factory%initialized) call current%factory%initialize()
2856         current => current%next
2857      end do
2858   end subroutine abstract_model_factory_initialize
2859
2860   subroutine abstract_model_factory_add(self, child, prefix)
2861      class (type_base_model_factory),         intent(inout) :: self
2862      class (type_base_model_factory), target, intent(in)    :: child
2863      character(len=*), optional,              intent(in)    :: prefix
2864
2865      type (type_base_model_factory_node), pointer :: current
2866
2867      if (self%initialized) call driver%fatal_error('abstract_model_factory_add', &
2868         'BUG! Factory initialiation is complete. Child factories can no longer be added.')
2869
2870      if (.not.associated(self%first_child)) then
2871         allocate(self%first_child)
2872         current => self%first_child
2873      else
2874         current => self%first_child
2875         do while(associated(current%next))
2876            current => current%next
2877         end do
2878         allocate(current%next)
2879         current => current%next
2880      end if
2881
2882      current%factory => child
2883      if (present(prefix)) current%prefix = prefix
2884   end subroutine abstract_model_factory_add
2885
2886   recursive subroutine abstract_model_factory_create(self, name, model)
2887      class (type_base_model_factory), intent(in) :: self
2888      character(len=*),                intent(in) :: name
2889      class (type_base_model), pointer            :: model
2890
2891      type (type_base_model_factory_node), pointer :: child
2892      integer                                      :: n
2893
2894      child => self%first_child
2895      do while(associated(child))
2896         if (child%prefix /= '') then
2897            n = len_trim(child%prefix)
2898            if (len_trim(name) > n + 1) then
2899               if (name(1:n) == child%prefix .and. (name(n + 1:n + 1) == '_' .or. name(n + 1:n + 1) == '/')) &
2900                  call child%factory%create(name(n+2:), model)
2901            end if
2902         else
2903            call child%factory%create(name, model)
2904         end if
2905         if (associated(model)) return
2906         child => child%next
2907      end do
2908   end subroutine abstract_model_factory_create
2909
2910   recursive subroutine abstract_model_factory_register_version(self, name, version_string)
2911      class (type_base_model_factory),intent(in) :: self
2912      character(len=*),               intent(in) :: name, version_string
2913
2914      type (type_version), pointer :: version
2915
2916      if (associated(first_module_version)) then
2917         version => first_module_version
2918         do while (associated(version%next))
2919            version => version%next
2920         end do
2921         allocate(version%next)
2922         version => version%next
2923      else
2924         allocate(first_module_version)
2925         version => first_module_version
2926      end if
2927      version%module_name = name
2928      version%version_string = version_string
2929   end subroutine abstract_model_factory_register_version
2930
2931   recursive subroutine abstract_model_factory_finalize(self)
2932      class (type_base_model_factory), intent(inout) :: self
2933
2934      type (type_base_model_factory_node), pointer :: current, next
2935
2936      current => self%first_child
2937      do while(associated(current))
2938         next => current%next
2939         call current%factory%finalize()
```

```
2940          deallocate(current)
2941          current => next
2942       end do
2943       self%first_child => null()
2944    end subroutine abstract_model_factory_finalize
2945
2946    subroutine coupling_task_list_remove(self, task)
2947       class (type_coupling_task_list), intent(inout) :: self
2948       class (type_coupling_task), pointer            :: task
2949       if (associated(task%previous)) then
2950          task%previous%next => task%next
2951       else
2952          self%first => task%next
2953       end if
2954       if (associated(task%next)) task%next%previous => task%previous
2955       deallocate(task)
2956    end subroutine
2957
2958    function coupling_task_list_add_object(self, task, always_create) result(used)
2959       class (type_coupling_task_list), intent(inout) :: self
2960       class (type_coupling_task), pointer            :: task
2961       logical,                         intent(in)    :: always_create
2962       logical                                        :: used
2963
2964       class (type_coupling_task), pointer :: existing_task
2965
2966       ! First try to find an existing coupling task for this link. If one exists, we'll replace it.
2967       used = .false.
2968       existing_task => self%first
2969       do while (associated(existing_task))
2970          ! Check if we have found an existing task for the same link.
2971          if (associated(existing_task%slave, task%slave)) then
2972             ! If existing one has higher priority, do not add the new task and return (used=.false.)
2973             if (existing_task%user_specified .and. .not. always_create) return
2974
2975             ! We will overwrite the existing task - remove existing task and exit loop
2976             call self%remove(existing_task)
2977             exit
2978          end if
2979          existing_task => existing_task%next
2980       end do
2981
2982       used = .true.
2983       if (.not. associated(self%first)) then
2984          ! Task list is empty - add first.
2985          self%first => task
2986          task%previous => null()
2987       else
2988          ! Task list contains items - append to tail.
2989
2990          ! Find tail of the list
2991          existing_task => self%first
2992          do while (associated(existing_task%next))
2993             existing_task => existing_task%next
2994          end do
2995
2996          existing_task%next => task
2997          task%previous => existing_task
2998       end if
2999       task%next => null()
3000    end function coupling_task_list_add_object
3001
3002    subroutine coupling_task_list_add(self, link, always_create, task)
3003       class (type_coupling_task_list), intent(inout)         :: self
3004       type (type_link),                intent(inout), target :: link
3005       logical,                         intent(in)            :: always_create
3006       class (type_coupling_task), pointer                    :: task
3007
3008       logical :: used
3009
3010       allocate(task)
3011       task%slave => link
3012       used = self%add_object(task, always_create)
3013       if (.not. used) deallocate(task)
3014    end subroutine coupling_task_list_add
3015
3016    character(len=32) function source2string(source)
3017       integer, intent(in) :: source
3018       select case (source)
3019       case (source_unknown);                  source2string = 'unknown'
3020       case (source_state);                    source2string = 'state'
3021       case (source_external);                 source2string = 'external'
3022       case (source_do);                       source2string = 'do'
3023       case (source_do_column);                source2string = 'do_column'
3024       case (source_do_horizontal);            source2string = 'do_horizontal'
3025       case (source_do_bottom);                source2string = 'do_bottom'
3026       case (source_do_surface);               source2string = 'do_surface'
3027       case (source_constant);                 source2string = 'constant'
3028       case (source_get_vertical_movement);    source2string = 'get_vertical_movement'
3029       case (source_check_state);              source2string = 'check_state'
3030       case (source_check_bottom_state);       source2string = 'check_bottom_state'
3031       case (source_check_surface_state);      source2string = 'check_surface_state'
3032       case (source_initialize_state);         source2string = 'initialize_state'
3033       case (source_initialize_bottom_state);  source2string = 'initialize_bottom_state'
3034       case (source_initialize_surface_state); source2string = 'initialize_surface_state'
3035       case (source_get_light_extinction);     source2string = 'get_light_extinction'
3036       case (source_get_drag);                 source2string = 'get_drag'
3037       case (source_get_albedo);               source2string = 'get_albedo'
```

```
3038        case default
3039           write (source2string,'(i0)') source
3040        end select
3041     end function source2string
3042
3043     subroutine variable_set_add(self, variable)
3044        class (type_variable_set), intent(inout) :: self
3045        type (type_internal_variable), target    :: variable
3046
3047        type (type_variable_node), pointer :: node
3048
3049        ! Check if this variable already exists.
3050        node => self%first
3051        do while (associated(node))
3052           if (associated(node%target, variable)) return
3053           node => node%next
3054        end do
3055
3056        ! Create a new variable object and prepend it to the list.
3057        allocate(node)
3058        node%target => variable
3059        node%next => self%first
3060        self%first => node
3061     end subroutine variable_set_add
3062
3063     subroutine variable_set_remove(self, variable, discard)
3064        class (type_variable_set), intent(inout) :: self
3065        type (type_internal_variable), target    :: variable
3066        logical, optional,         intent(in)    :: discard
3067
3068        type (type_variable_node), pointer :: node, previous
3069        logical                            :: discard_
3070
3071        ! Check if this variable already exists.
3072        previous => null()
3073        node => self%first
3074        do while (associated(node))
3075           if (associated(node%target, variable)) then
3076              if (associated(previous)) then
3077                 previous%next => node%next
3078              else
3079                 self%first => node%next
3080              end if
3081              deallocate(node)
3082              return
3083           end if
3084           previous => node
3085           node => node%next
3086        end do
3087        discard_ = .false.
3088        if (present(discard)) discard_ = discard
3089        if (.not. discard_) call driver%fatal_error('variable_set_remove', &
3090           'Variable "' // trim(variable%name) // '" not found in set.')
3091     end subroutine variable_set_remove
3092
3093     logical function variable_set_contains(self, variable)
3094        class (type_variable_set), intent(in) :: self
3095        type (type_internal_variable), target :: variable
3096
3097        type (type_variable_node), pointer :: node
3098
3099        variable_set_contains = .true.
3100        node => self%first
3101        do while (associated(node))
3102           if (associated(node%target, variable)) return
3103           node => node%next
3104        end do
3105        variable_set_contains = .false.
3106     end function variable_set_contains
3107
3108     subroutine variable_set_update(self, other)
3109        class (type_variable_set), intent(inout) :: self
3110        class (type_variable_set), intent(in)    :: other
3111
3112        type (type_variable_node), pointer :: node
3113
3114        node => other%first
3115        do while (associated(node))
3116           call self%add(node%target)
3117           node => node%next
3118        end do
3119     end subroutine variable_set_update
3120
3121     subroutine variable_set_finalize(self)
3122        class (type_variable_set), intent(inout) :: self
3123
3124        type (type_variable_node), pointer :: node, next
3125
3126        node => self%first
3127        do while (associated(node))
3128           next => node%next
3129           deallocate(node)
3130           node => next
3131        end do
3132        self%first => null()
3133     end subroutine variable_set_finalize
3134
3135     subroutine variable_list_append(self, variable, index)
```

```
      class (type_variable_list), intent(inout) :: self
      type (type_internal_variable), target     :: variable
      integer,optional,           intent(out)   :: index

      type (type_variable_node), pointer :: last

      if (associated(self%first)) then
         last => self%first
         do while (associated(last%next))
            last => last%next
         end do
         allocate(last%next)
         last%next%target => variable
      else
         allocate(self%first)
         self%first%target => variable
      end if
      self%count = self%count + 1
      if (present(index)) index = self%count
   end subroutine variable_list_append

   subroutine variable_list_finalize(self)
      class (type_variable_list), intent(inout) :: self

      type (type_variable_node), pointer :: node, next

      node => self%first
      do while (associated(node))
         next => node%next
         deallocate(node)
         node => next
      end do
      self%first => null()
      self%count = 0
   end subroutine

end module fabm_types

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