```fortran
#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

   type, extends(type_variable_id) :: type_horizontal_dependency_id
      integer  :: horizontal_index = -1
      real(rk) :: background       = 0.0_rk
   end type

   type, extends(type_horizontal_dependency_id) :: type_bottom_dependency_id
   end type

   type, extends(type_horizontal_dependency_id) :: type_surface_dependency_id
   end type

   type, extends(type_variable_id) :: type_global_dependency_id
      integer  :: global_index = -1
      real(rk) :: background   = 0.0_rk
   end type

   type, extends(type_dependency_id) :: type_state_variable_id
      integer :: state_index = -1
      type (type_add_id)            :: sms
      type (type_add_id)            :: movement
      type (type_horizontal_add_id) :: surface_flux
      type (type_horizontal_add_id) :: bottom_flux
   end type

   type, extends(type_bottom_dependency_id) :: type_bottom_state_variable_id
      integer :: bottom_state_index = -1
      type (type_horizontal_add_id) :: bottom_sms
   end type

   type, extends(type_surface_dependency_id) :: type_surface_state_variable_id
      integer :: surface_state_index = -1
      type (type_horizontal_add_id) :: surface_sms
   end type

   type, extends(type_variable_id) :: type_diagnostic_variable_id
      integer :: write_index = -1
   end type

   type, extends(type_variable_id) :: type_surface_diagnostic_variable_id
      integer :: surface_write_index = -1
   end type

   type, extends(type_variable_id) :: type_bottom_diagnostic_variable_id
      integer :: bottom_write_index = -1
   end type

   type, extends(type_variable_id) :: type_horizontal_diagnostic_variable_id
      integer :: horizontal_write_index = -1
   end type

   ! --------------------------------------------------------------------------
   ! Data types for contributions to aggregate variables.
   ! --------------------------------------------------------------------------

   type type_contribution
      class (type_domain_specific_standard_variable), pointer :: target => null()
      real(rk)                                        :: scale_factor = 1.0_rk
      logical                                         :: include_background = .false.
      type (type_contribution),            pointer :: next => null()
   end type

   type type_contribution_list
      type (type_contribution), pointer :: first => null()
   contains
      procedure :: add      => contribution_list_add
      procedure :: finalize => contribution_list_finalize
   end type

   type type_aggregate_variable_access
      class (type_domain_specific_standard_variable), pointer :: standard_variable => null()
      integer                                              :: access            = access_none
      type (type_aggregate_variable_access),          pointer :: next              => null()
   end type

   ! --------------------------------------------------------------------------
   ! Data types for collections of variables
   ! --------------------------------------------------------------------------

   type type_link_list
      type (type_link), pointer :: first => null()
      type (type_link), pointer :: last  => null()
   contains
      procedure :: append   => link_list_append
      procedure :: find     => link_list_find
      procedure :: count    => link_list_count
      procedure :: extend   => link_list_extend
      procedure :: finalize => link_list_finalize
   end type

   type type_link_pointer
      type (type_link),         pointer :: p    => null()
      type (type_link_pointer), pointer :: next => null()
   end type

   type type_variable_node
      type (type_internal_variable), pointer :: target => null()
      type (type_variable_node),     pointer :: next   => null()
   end type

   type type_variable_set
      type (type_variable_node), pointer :: first => null()
   contains
      procedure :: add      => variable_set_add
      procedure :: update   => variable_set_update
      procedure :: remove   => variable_set_remove
      procedure :: contains => variable_set_contains
      procedure :: finalize => variable_set_finalize
   end type

   type type_variable_list
      type (type_variable_node), pointer :: first => null()
      integer                            :: count = 0
   contains
      procedure :: append   => variable_list_append
      procedure :: finalize => variable_list_finalize
   end type

   ! --------------------------------------------------------------------------
   ! Data types for information on model variables and model references
   ! --------------------------------------------------------------------------

   type type_internal_variable
      character(len=attribute_length) :: name      = ''
      character(len=attribute_length) :: long_name = ''
      type (type_property_dictionary) :: properties
      character(len=attribute_length) :: units          = ''
      real(rk)                        :: minimum        = -1.e20_rk
      real(rk)                        :: maximum        =  1.e20_rk
      real(rk)                        :: missing_value  = -2.e20_rk
      real(rk)                        :: prefill_value  = -2.e20_rk
      real(rk)                        :: initial_value  = 0.0_rk
      integer                         :: output         = output_instantaneous
      integer                         :: presence       = presence_internal
      integer                         :: domain         = domain_interior
      integer                         :: source         = source_unknown
      integer                         :: prefill        = prefill_none
      integer                         :: write_operator = operator_assign
      class (type_base_model),pointer :: owner          => null()
      type (type_contribution_list)   :: contributions

      type (type_standard_variable_set) :: standard_variables

      logical :: fake_state_variable = .false.

      ! Only used for interior state variables:
      logical :: no_precipitation_dilution = .false.
      logical :: no_river_dilution         = .false.

      integer, pointer :: read_index  => null()
      integer, pointer :: write_index => null()
      integer          :: store_index = store_index_none
      integer          :: catalog_index = -1

      ! Collections to collect information from all coupled variables.
      type (type_integer_pointer_set)  :: read_indices, state_indices, write_indices
      type (type_real_pointer_set)     :: background_values
      type (type_link_list)            :: sms_list, surface_flux_list, bottom_flux_list, movement_list
      type (type_link), pointer        :: sms_sum            => null()
      type (type_link), pointer        :: surface_flux_sum    => null()
      type (type_link), pointer        :: bottom_flux_sum     => null()
      type (type_link), pointer        :: movement_sum        => null()
      type (type_link), pointer        :: sms                 => null()
      type (type_link), pointer        :: surface_flux        => null()
      type (type_link), pointer        :: bottom_flux         => null()

      type (type_internal_variable), pointer :: write_owner => null()
      type (type_variable_set),      pointer :: cowriters   => null()
      type (type_link_pointer),      pointer :: first_link  => null()
   end type

   type type_link
      character(len=attribute_length)        :: name     = ''
      type (type_internal_variable), pointer :: target   => null()
      type (type_internal_variable), pointer :: original => null()
      type (type_link), pointer              :: next     => null()
   end type

   ! --------------------------------------------------------------------------
   ! Data type for custom expressions (arbitrary functions of one or more
   ! variables).
   ! --------------------------------------------------------------------------

   type, abstract :: type_expression
      class (type_expression), pointer :: next        => null()
      character(len=attribute_length)  :: output_name = ''
      integer, pointer :: out => null()
   end type

   type, abstract, extends(type_expression) :: type_interior_expression
      !type (type_interior_data_pointer), pointer :: out => null()
   end type

   type, abstract, extends(type_expression) :: type_horizontal_expression
      !type (type_horizontal_data_pointer), pointer :: out => null()
   end type
```

```fortran

   ! --------------------------------------------------------------------------
   ! Data type for collection of models
   ! --------------------------------------------------------------------------

   type type_model_list_node
      class (type_base_model),     pointer :: model => null()
      type (type_model_list_node), pointer :: next  => null()
   end type

   type type_model_list
      type (type_model_list_node), pointer :: first => null()
   contains
      procedure :: append     => model_list_append
      procedure :: extend     => model_list_extend
      procedure :: find_name  => model_list_find_name
      procedure :: find_model => model_list_find_model
      procedure :: count      => model_list_count
      procedure :: finalize   => model_list_finalize
      procedure :: print      => model_list_print
      generic   :: find       => find_name, find_model
   end type

   ! --------------------------------------------------------------------------
   ! Base model type, used by biogeochemical models to inherit from, and by
   ! external host to get variable lists and metadata.
   ! --------------------------------------------------------------------------

   type type_base_model
      ! Flag determining whether the contents of the type are "frozen", i.e., they will not change anymore.
      logical :: frozen = .false.

      ! Flag determining whether this model was explicitly created by the user (by it appearing as instance in fabm.y
aml)
      logical :: user_created = .false.

      ! Pointers to linked models in the model tree.
      class (type_base_model), pointer :: parent => null()
      type (type_model_list)           :: children

      ! Model name and variable prefixes.
      character(len=attribute_length) :: name      = ''
      character(len=attribute_length) :: long_name = ''
      character(len=attribute_length) :: type_name = ''

      ! Models constituents: links to variables, coupling requests, parameters, expressions
      type (type_link_list) :: links
      type (type_aggregate_variable_access), pointer :: first_aggregate_variable_access => null()

      type (type_hierarchical_dictionary) :: couplings
      type (type_hierarchical_dictionary) :: parameters

      class (type_expression), pointer :: first_expression => null()

      type (type_coupling_task_list) :: coupling_task_list

      real(rk) :: dt = 1.0_rk
      real(rk) :: rdt__ = 1.0_rk

      logical :: check_conservation = .false.

      type (type_add_id)            :: extinction_id
      type (type_horizontal_add_id) :: albedo_id
      type (type_horizontal_add_id) :: surface_drag_id

      integer, allocatable :: implemented(:)
   contains

      ! Procedure for adding child models [during initialization only]
      procedure :: add_child

      ! Procedures for adding variables [during initialization only]
      procedure :: add_interior_variable
      procedure :: add_horizontal_variable
      procedure :: add_scalar_variable
      procedure :: add_object

      ! Procedures for locating links, objects, models.
      procedure :: find_link
      procedure :: find_object
      procedure :: find_model

      ! Procedures for requesting coupling between variables
      procedure :: request_coupling_for_link
      procedure :: request_coupling_for_name
      procedure :: request_coupling_for_id
      procedure :: request_standard_coupling_for_link
      procedure :: request_standard_coupling_for_id
      generic   :: request_coupling => request_coupling_for_link, request_coupling_for_name, request_coupling_for_id,
 &
                                       request_standard_coupling_for_link, request_standard_coupling_for_id

      ! Procedures that may be used to query parameter values during initialization.
      procedure :: get_real_parameter
      procedure :: get_integer_parameter
      procedure :: get_logical_parameter
      procedure :: get_string_parameter
      generic :: get_parameter => get_real_parameter,get_integer_parameter,get_logical_parameter,get_string_parameter

      procedure :: set_variable_property_real
      procedure :: set_variable_property_integer
      procedure :: set_variable_property_logical
      generic   :: set_variable_property => set_variable_property_real,set_variable_property_integer,set_variable_property_logical

      procedure :: add_variable_to_aggregate_variable
      procedure :: add_constant_to_aggregate_variable
      generic :: add_to_aggregate_variable => add_variable_to_aggregate_variable, &
                                              add_constant_to_aggregate_variable

      ! Procedures that may be used to register model variables and dependencies during initialization.
      procedure :: register_source
      procedure :: register_surface_flux
      procedure :: register_bottom_flux
      procedure :: register_surface_source
      procedure :: register_bottom_source

      procedure :: register_interior_state_variable
      procedure :: register_bottom_state_variable
      procedure :: register_surface_state_variable

      procedure :: register_interior_diagnostic_variable
      procedure :: register_surface_diagnostic_variable
      procedure :: register_bottom_diagnostic_variable
      procedure :: register_horizontal_diagnostic_variable

      procedure :: register_named_interior_dependency
      procedure :: register_standard_interior_dependency
      procedure :: register_universal_interior_dependency
      procedure :: register_named_horizontal_dependency
      procedure :: register_standard_horizontal_dependency
      procedure :: register_standard_horizontal_dependency2
      procedure :: register_standard_horizontal_dependency3
      procedure :: register_universal_horizontal_dependency
      procedure :: register_named_surface_dependency
      procedure :: register_standard_surface_dependency
      procedure :: register_standard_surface_dependency2
      procedure :: register_universal_surface_dependency
      procedure :: register_named_bottom_dependency
      procedure :: register_standard_bottom_dependency
      procedure :: register_standard_bottom_dependency2
      procedure :: register_universal_bottom_dependency
      procedure :: register_named_global_dependency
      procedure :: register_standard_global_dependency

      generic :: register_interior_dependency   => register_named_interior_dependency, register_standard_interior_dependency, &
                                                   register_universal_interior_dependency
      generic :: register_horizontal_dependency => register_named_horizontal_dependency, register_standard_horizontal_dependency, &
                                                   register_standard_horizontal_dependency2, register_standard_horizontal_dependency3, &
                                                   register_universal_horizontal_dependency
      generic :: register_surface_dependency    => register_named_surface_dependency, register_standard_surface_dependency, &
                                                   register_standard_surface_dependency2, register_universal_surface_dependency
      generic :: register_bottom_dependency     => register_named_bottom_dependency, register_standard_bottom_dependency, &
                                                   register_standard_bottom_dependency2, register_universal_bottom_dependency
      generic :: register_global_dependency     => register_named_global_dependency, register_standard_global_dependency

      procedure :: register_interior_state_dependency
      procedure :: register_bottom_state_dependency
      procedure :: register_surface_state_dependency
      procedure :: register_standard_interior_state_dependency
      procedure :: register_standard_bottom_state_dependency
      procedure :: register_standard_bottom_state_dependency2
      procedure :: register_standard_surface_state_dependency
      procedure :: register_standard_surface_state_dependency2

      procedure :: register_interior_expression_dependency
      procedure :: register_horizontal_expression_dependency
      generic :: register_expression_dependency => register_interior_expression_dependency, register_horizontal_expression_dependency

      generic :: register_state_variable      => register_interior_state_variable, register_bottom_state_variable, &
                                                 register_surface_state_variable
      generic :: register_diagnostic_variable => register_interior_diagnostic_variable, register_horizontal_diagnostic_variable, &
                                                 register_surface_diagnostic_variable, register_bottom_diagnostic_variable
      generic :: register_dependency          => register_named_interior_dependency, register_standard_interior_dependency, &
                                                 register_universal_interior_dependency, &
                                                 register_named_horizontal_dependency, register_standard_horizontal_dependency, &
                                                 register_standard_horizontal_dependency2, register_standard_horizontal_dependency3, &
                                                 register_universal_horizontal_dependency, &
                                                 register_named_surface_dependency, register_standard_surface_dependency, &
                                                 register_standard_surface_dependency2, register_universal_surface_de
```

```
pendency, &
                                                 register_named_bottom_dependency, register_standard_bottom_dependency, &
                                                 register_standard_bottom_dependency2, register_universal_bottom_dependency, &
                                                 register_named_global_dependency, register_standard_global_dependency, &
                                                 register_interior_expression_dependency, register_horizontal_expression_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
```

```fortran
      ! Write cache (separate interior, horizontal fields).
      real(rk), allocatable _ĐIMENSION_SLICE_PLUS_1_            :: write
      real(rk), allocatable _ĐIMENSION_HORIZONTAL_SLICE_PLUS_1_ :: write_hz
   end type

   ! ====================================================================================================
   ! Base type for a model object factory (generates a model object from a model name)
   ! An implementation of this type is provided in fabm_library.F90.
   ! Institutes or groups can create inherit from this type to create their own model factories,
   ! which then need to be added to the root factory in fabm_library.F90.
   ! This makes it possible to introduce a large number of new models with only two lines added
   ! in the FABM core.
   ! ====================================================================================================

   type, public :: type_version
      character(len=attribute_length) :: module_name    = ''
      character(len=attribute_length) :: version_string = ''
      type (type_version), pointer    :: next           => null()
   end type
   type (type_version), pointer, save, public :: first_module_version => null()

   type type_base_model_factory_node
      character(len=attribute_length)              :: prefix  = ''
      class (type_base_model_factory),     pointer :: factory => null()
      type (type_base_model_factory_node), pointer :: next    => null()
   end type

   type, public :: type_base_model_factory
      type (type_base_model_factory_node), pointer :: first_child => null()
      logical                                      :: initialized = .false.
   contains
      procedure :: initialize       => abstract_model_factory_initialize
      procedure :: add              => abstract_model_factory_add
      procedure :: create           => abstract_model_factory_create
      procedure :: register_version => abstract_model_factory_register_version
      procedure :: finalize         => abstract_model_factory_finalize
   end type

   class (type_base_model_factory), pointer, save, public :: factory => null()

contains

   subroutine base_initialize(self, configunit)
      class (type_base_model), intent(inout), target :: self
      integer,                 intent(in)            :: configunit
   end subroutine

   subroutine base_initialize_state(self, _ARGUMENTS_INITIALIZE_STATE_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_INITIALIZE_STATE_
   end subroutine

   subroutine base_initialize_horizontal_state(self, _ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_
   end subroutine

   ! Providing process rates and diagnostics
   subroutine base_do(self, _ARGUMENTS_ĐO_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_
   end subroutine

   subroutine base_do_ppdd(self, _ARGUMENTS_ĐO_PPĐĐ_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_PPĐĐ_
      call self%do(_ARGUMENTS_ĐO_)
   end subroutine

   subroutine base_do_bottom(self, _ARGUMENTS_ĐO_BOTTOM_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_BOTTOM_
   end subroutine

   subroutine base_do_bottom_ppdd(self, _ARGUMENTS_ĐO_BOTTOM_PPĐĐ_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_BOTTOM_PPĐĐ_
   end subroutine

   subroutine base_do_surface(self, _ARGUMENTS_ĐO_SURFACE_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_SURFACE_
   end subroutine

   subroutine base_do_horizontal(self, _ARGUMENTS_HORIZONTAL_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_HORIZONTAL_
   end subroutine

   subroutine base_do_column(self, _ARGUMENTS_ĐO_COLUMN_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_ARGUMENTS_ĐO_COLUMN_
      call self%get_light(_ARGUMENTS_ĐO_COLUMN_)
   end subroutine

   subroutine base_get_vertical_movement(self, _ARGUMENTS_GET_VERTICAL_MOVEMENT_)
      class (type_base_model), intent(in) :: self
      _ĐECLARE_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

   subroutine base_get_drag(self, _ARGUMENTS_GET_ÐRAG_)
      class (type_base_model), intent(in) :: self
      _ÐECLARE_ARGUMENTS_GET_ÐRAG_
   end subroutine

   subroutine base_get_albedo(self, _ARGUMENTS_GET_ALBEÐO_)
      class (type_base_model), intent(in) :: self
      _ÐECLARE_ARGUMENTS_GET_ALBEÐO_
   end subroutine

   subroutine base_get_light(self, _ARGUMENTS_ÐO_COLUMN_)
      class (type_base_model), intent(in) :: self
      _ÐECLARE_ARGUMENTS_ÐO_COLUMN_
   end subroutine

   function base_get_path(self) result(path)
      class (type_base_model), intent(in), target :: self
      character(len=attribute_length)             :: path

      class (type_base_model), pointer :: current

      path = ''
      current => self
      do while (associated(current%parent))
         path = '/' // trim(current%name) // trim(path)
         current => current%parent
      end do
   end function

   subroutine base_fatal_error(self, location, message)
      class (type_base_model), intent(in) :: self
      character(len=*),        intent(in) :: location, message
      if (self%name /= '') then
         call driver%fatal_error('model ' // trim(self%get_path()) // ', ' // trim(location), message)
      else
         call driver%fatal_error(location, message)
      end if
   end subroutine

   subroutine base_log_message(self, message)
      class (type_base_model), intent(in) :: self
      character(len=*),        intent(in) :: message
      if (self%name /= '') then
         call driver%log_message('model "' // trim(self%name) // '": ' // message)
      else
         call driver%log_message(message)
      end if
   end subroutine

   subroutine base_before_coupling(self)
      class (type_base_model), intent(inout) :: self
   end subroutine

   subroutine base_after_coupling(self)
      class (type_base_model), intent(inout) :: self
   end subroutine

   function implements(self, source) result(is_implemented)
      class (type_base_model), intent(in) :: self
      integer,                 intent(in) :: source
      logical                             :: is_implemented

      integer :: i

      is_implemented = .true.
      if (allocated(self%implemented)) then
         do i = 1, size(self%implemented)
            if (self%implemented(i) == source) return
         end do
         is_implemented = .false.
      end if
   end function

   subroutine register_implemented_routines(self, sources)
      class (type_base_model), intent(inout) :: self
      integer, optional,       intent(in)    :: sources(:)
      if (allocated(self%implemented)) deallocate(self%implemented)
      if (present(sources)) then
         allocate(self%implemented(size(sources)))
         self%implemented(:) = sources
      else
         allocate(self%implemented(0))
      end if
   end subroutine

   recursive subroutine add_child(self, model, name, long_name, configunit)
      class (type_base_model), target, intent(inout) :: self, model
      character(len=*),                intent(in)    :: name
      character(len=*), optional,      intent(in)    :: long_name
      integer, optional,               intent(in)    :: configunit

      integer                                :: islash
      class (type_base_model),     pointer :: parent
      type (type_model_list_node), pointer :: child
      integer                                :: ind

      ! If a path with / is given, redirect to tentative parent model.
```

```fortran
      islash = index(name, '/', .true.)
      if (islash /= 0) then
         parent => self%find_model(name(:islash - 1))
         if (.not. associated(parent)) call self%fatal_error('add_child', &
            'Proposed parent model "' // trim(name(:islash - 1)) // '" was not found.')
         call parent%add_child(model, name(islash + 1:), long_name, configunit)
         return
      end if

      if (associated(model%parent)) call self%fatal_error('add_child', &
         'The provided child model "' // trim(name) // '" has already been assigned parent ' // trim(model%parent%name) // '.')

      if (name == '*') then
         ! This instance is for internal use only - auto-generate a unique name
         ind = 1
         do
            write (model%name, '("_", i0)') ind
            child => self%children%first
            do while (associated(child))
               if (child%model%name == model%name) exit
               child => child%next
            end do
            if (.not. associated(child)) exit
            ind = ind + 1
         end do
      else
         ! Ascertain whether the provided name is valid.
         if (name == '') call self%fatal_error('add_child', 'Invalid model name "' // trim(name) // &
            '". Names cannot be empty.')
         if (name(1:1) == '_') call self%fatal_error('add_child', 'Invalid model name "' // trim(name) // &
            '". Names beginning with underscore are reserved for internal use.')
         if (len_trim(name) > len(model%name)) call self%fatal_error('add_child', 'Invalid model name "' // trim(name) // &
            '". This name is longer than the maximum allowed number of characters.')
         if (name /= get_safe_name(name)) call self%fatal_error('add_child', 'Invalid model name " '// trim(name) // &
            '". Names can contain letters, digits and underscores only.')

         ! Make sure a child with this name does not exist yet.
         child => self%children%first
         do while (associated(child))
            if (child%model%name == name) call self%fatal_error('add_child', &
               'A child model with name "' // trim(name) // '" already exists.')
            child => child%next
         end do
         model%name = name
      end if

      if (present(long_name)) then
         model%long_name = trim(long_name)
      else
         model%long_name = trim(model%name)
      end if
      model%parent => self
      call self%parameters%add_child(model%parameters, trim(model%name))
      call self%couplings%add_child(model%couplings, trim(model%name))
      call self%children%append(model)
      call model%initialize(-1)
      model%rdt__ = 1._rk / model%dt

      if (model%implements(source_get_light_extinction)) then
         call model%add_interior_variable('_attenuation_coefficient_of_photosynthetic_radiative_flux', 'm-1', &
            'light extinction contribution computed by get_light_extinction', fill_value=0.0_rk, missing_value=0.0_rk, &
            output=output_none, write_index=model%extinction_id%sum_index, link=model%extinction_id%link, &
            source=source_get_light_extinction)
         model%extinction_id%link%target%write_operator = operator_add
         call model%add_to_aggregate_variable(standard_variables%attenuation_coefficient_of_photosynthetic_radiative_flux, &
            model%extinction_id)
      end if

      if (model%implements(source_get_albedo)) then
         call model%add_horizontal_variable('_surface_albedo', '-', &
            'albedo contribution computed by get_albedo', fill_value=0.0_rk, missing_value=0.0_rk, &
            output=output_none, write_index=model%albedo_id%horizontal_sum_index, link=model%albedo_id%link, &
            source=source_get_albedo)
         model%albedo_id%link%target%write_operator = operator_add
         call model%add_to_aggregate_variable(standard_variables%surface_albedo, model%albedo_id)
      end if

      if (model%implements(source_get_drag)) then
         call model%add_horizontal_variable('_surface_drag_coefficient_in_air', '-', &
            'surface drag contribution computed by get_drag', fill_value=0.0_rk, missing_value=0.0_rk, &
            output=output_none, write_index=model%surface_drag_id%horizontal_sum_index, link=model%surface_drag_id%link, &
            source=source_get_drag)
         model%surface_drag_id%link%target%write_operator = operator_add
         call model%add_to_aggregate_variable(standard_variables%surface_drag_coefficient_in_air, model%surface_drag_id)
      end if
   end subroutine add_child

   subroutine set_variable_property_real(self, variable, name, value)
      class (type_base_model),  intent(inout) :: self
      class (type_variable_id), intent(inout) :: variable
      character(len=*),         intent(in)    :: name
      real(rk),                  intent(in)    :: value
      if (.not. associated(variable%link)) call self%fatal_error('set_variable_property_real', 'variable has not been registered')
      call variable%link%target%properties%set_real(name, value)
   end subroutine

   subroutine set_variable_property_integer(self, variable, name, value)
      class (type_base_model),  intent(inout) :: self
      class (type_variable_id), intent(inout) :: variable
      character(len=*),         intent(in)    :: name
      integer,                  intent(in)    :: value
      if (.not. associated(variable%link)) call self%fatal_error('set_variable_property_integer', 'variable has not been registered')
      call variable%link%target%properties%set_integer(name, value)
   end subroutine

   subroutine set_variable_property_logical(self, variable, name, value)
      class (type_base_model), intent(inout) :: self
      class (type_variable_id),intent(inout) :: variable
      character(len=*),        intent(in)    :: name
      logical,                 intent(in)    :: value
      if (.not.associated(variable%link)) call self%fatal_error('set_variable_property_logical', 'variable has not been registered')
      call variable%link%target%properties%set_logical(name, value)
   end subroutine

   subroutine add_variable_to_aggregate_variable(self, target, variable_id, scale_factor, include_background)
      use fabm_standard_variables   ! workaround for bug in Cray compiler 8.3.7
      class (type_base_model),            intent(inout) :: self
      class (type_base_standard_variable), intent(in)    :: target
      class (type_variable_id),           intent(inout) :: variable_id
      real(rk), optional,                 intent(in)    :: scale_factor
      logical, optional,                  intent(in)    :: include_background

      class (type_base_standard_variable), pointer :: standard_variable

      if (.not. target%aggregate_variable) call self%fatal_error('add_variable_to_aggregate_variable', &
            'target "' // trim(target%name) // '" is not an aggregate variable.')
      if (.not. associated(variable_id%link)) call self%fatal_error('add_to_aggregate_variable', &
         'variable added to ' // trim(target%name) // ' has not been registered')
      standard_variable => target%resolve()
      select type (standard_variable)
      class is (type_universal_standard_variable)
         select case(variable_id%link%target%domain)
         case (domain_interior)
            call variable_id%link%target%contributions%add(standard_variable%in_interior(), scale_factor, include_background)
         case (domain_horizontal)
            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_background)
         case (domain_surface)
            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_background)
            call variable_id%link%target%contributions%add(standard_variable%at_surface(), scale_factor, include_background)
         case (domain_bottom)
            call variable_id%link%target%contributions%add(standard_variable%at_interfaces(), scale_factor, include_background)
            call variable_id%link%target%contributions%add(standard_variable%at_bottom(), scale_factor, include_background)
         end select
      class is (type_domain_specific_standard_variable)
         call variable_id%link%target%contributions%add(standard_variable, scale_factor, include_background)
      end select
   end subroutine add_variable_to_aggregate_variable

   subroutine add_constant_to_aggregate_variable(self, target, value)
      class (type_base_model),                       intent(inout) :: self
      class (type_domain_specific_standard_variable), intent(in)    :: target
      real(rk),                                      intent(in)    :: value

      class (type_domain_specific_standard_variable), pointer :: standard_variable
      type (type_link),                              pointer :: link

      if (.not. target%aggregate_variable) call self%fatal_error('add_constant_to_aggregate_variable', &
           'target "' // trim(target%name) // '" is not an aggregate variable.')
      standard_variable => target%typed_resolve()

      link => null()
      select type (standard_variable)
      class is (type_interior_standard_variable)
         call self%add_interior_variable('_constant_*', standard_variable%units, standard_variable%name, source=source_constant, &
            fill_value=value, output=output_none, link=link)
         call link%target%contributions%add(standard_variable)
      class is (type_surface_standard_variable)
         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=source_constant, &
            fill_value=value, domain=domain_surface, output=output_none, link=link)
      class is (type_bottom_standard_variable)
         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=source_constant, &
            fill_value=value, domain=domain_bottom, output=output_none, link=link)
      class is (type_horizontal_standard_variable)
         call self%add_horizontal_variable('_constant_*', standard_variable%units, standard_variable%name, source=source_constant, &
            fill_value=value, output=output_none, link=link)
      end select
      call link%target%contributions%add(standard_variable)
   end subroutine add_constant_to_aggregate_variable

   subroutine contribution_list_add(self, standard_variable, scale_factor, include_background)
      class (type_contribution_list),      intent(inout) :: self
      class (type_domain_specific_standard_variable), target :: standard_variable
      real(rk), optional,                  intent(in)    :: scale_factor
      logical, optional,                   intent(in)    :: include_background

      type (type_contribution), pointer :: contribution
      logical,                  pointer :: pmember

      ! If the scale factor is 0, no need to register any contribution.
      if (present(scale_factor)) then
         if (scale_factor == 0.0_rk) return
      end if

      ! First look for existing contribution to this aggregate variable.
      contribution => self%first
      pmember => standard_variable%aggregate_variable
      do while (associated(contribution))
         ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
erences.
         if (associated(pmember, contribution%target%aggregate_variable)) exit
         contribution => contribution%next
      end do

      if (.not. associated(contribution)) then
         ! No contribution to this aggregate variable exists - prepend it to the list.
         allocate(contribution)
         contribution%next => self%first
         self%first => contribution
      end if

      ! Store contribution attributes
      contribution%target => standard_variable
      if (present(scale_factor)) contribution%scale_factor = scale_factor
      if (present(include_background)) contribution%include_background = include_background
   end subroutine

   subroutine contribution_list_finalize(self)
      class (type_contribution_list), intent(inout) :: self

      type (type_contribution), pointer :: contribution, next_contribution

      contribution => self%first
      do while (associated(contribution))
         next_contribution => contribution%next
         deallocate(contribution)
         contribution => next_contribution
      end do
      self%first => null()
   end subroutine

   subroutine model_list_append(self, model)
      class (type_model_list), intent(inout) :: self
      class (type_base_model), target        :: model

      type (type_model_list_node), pointer :: node

      if (.not. associated(self%first)) then
         allocate(self%first)
         node => self%first
      else
         node => self%first
         do while (associated(node%next))
            node => node%next
         end do
         allocate(node%next)
         node => node%next
      end if
      node%model => model
   end subroutine

   subroutine model_list_extend(self, source)
      class (type_model_list), intent(inout) :: self
      class (type_model_list), intent(in)    :: source

      type (type_model_list_node), pointer :: node

      node => source%first
      do while (associated(node))
         call self%append(node%model)
         node => node%next
      end do
   end subroutine

   function model_list_find_name(self, name) result(node)
      class (type_model_list), intent(in) :: self
      character(len=*),        intent(in) :: name

      type (type_model_list_node), pointer :: node

      node => self%first
      do while (associated(node))
         if (node%model%name == name) return
         node => node%next
      end do
   end function model_list_find_name

   function model_list_find_model(self, model) result(node)
      class (type_model_list),         intent(in) :: self
      class (type_base_model), target, intent(in) :: model

      type (type_model_list_node), pointer :: node
      logical,                     pointer :: pmember

      node => self%first
      pmember => model%frozen
      do while (associated(node))
         ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
erences.
         if (associated(pmember, node%model%frozen)) return
         node => node%next
      end do
   end function model_list_find_model

   subroutine model_list_print(self)
      class (type_model_list),       intent(in) :: self

      type (type_model_list_node),pointer :: node

      node => self%first
      do while (associated(node))
         call driver%log_message(node%model%get_path())
         node => node%next
      end do
   end subroutine

   function model_list_count(self, model) result(count)
      class (type_model_list),         intent(in) :: self
      class (type_base_model), target, intent(in) :: model

      integer :: count

      type (type_model_list_node), pointer :: node
      logical,                     pointer :: pmember

      count = 0
      node => self%first
      pmember => model%frozen
      do while (associated(node))
         ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
erences.
         if (associated(pmember, node%model%frozen)) count = count + 1
         node => node%next
      end do
   end function

   subroutine model_list_finalize(self)
      class (type_model_list), intent(inout) :: self

      type (type_model_list_node), pointer :: node, next

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

   function link_list_find(self, name) result(link)
      class (type_link_list), intent(in) :: self
      character(len=*),       intent(in) :: name

      type (type_link), pointer :: link

      link => self%first
      do while (associated(link))
         if (link%name == name) return
         link => link%next
      end do
   end function link_list_find

   function link_list_append(self, target, name) result(link)
      class (type_link_list),  intent(inout) :: self
      type (type_internal_variable), pointer :: target
      character(len=*),        intent(in)    :: name

      type (type_link), pointer :: link

      ! Append a new link to the list.
      if (.not. associated(self%first)) then
         allocate(self%first)
         self%last => self%first
      else
         allocate(self%last%next)
         self%last => self%last%next
      end if

      ! Set link attributes.
      link => self%last
      link%name = name
      link%target => target
      link%original => target
   end function link_list_append

   subroutine link_list_extend(self, source)
      class (type_link_list), intent(inout) :: self
      class (type_link_list), intent(in)    :: source

      type (type_link), pointer :: source_link, link

      source_link => source%first
      do while (associated(source_link))
         link => self%append(source_link%target, source_link%name)
         source_link => source_link%next
      end do
   end subroutine link_list_extend

   function link_list_count(self) result(count)
      class (type_link_list), intent(in) :: self
      integer                            :: count

      type (type_link), pointer :: link

      count = 0
      link => self%first
      do while (associated(link))
         count = count + 1
         link => link%next
      end do
   end function link_list_count

   subroutine link_list_finalize(self)
      class (type_link_list), intent(inout) :: self

      type (type_link), pointer :: link, next

      link => self%first
      do while (associated(link))
         next => link%next
         deallocate(link)
         link => next
      end do
      self%first => null()
   end subroutine link_list_finalize

   subroutine create_coupling_task(self, link, task)
      class (type_base_model),  intent(inout) :: self
      type (type_link), target, intent(inout) :: link
      class (type_coupling_task), pointer     :: task

      type (type_link), pointer :: current_link

      ! First make sure that we are called for a link that we own ourselves.
      current_link => self%links%first
      do while (associated(current_link))
         if (associated(current_link, link)) exit
         current_link => current_link%next
      end do
      if (.not.associated(current_link)) call self%fatal_error('request_coupling_for_link', &
         'Couplings can only be requested for variables that you own yourself.')

      ! Make sure that the link also points to a variable that we registered ourselves,
      ! rather than one registered by a child model.
      if (index(link%name, '/') /= 0) call self%fatal_error('request_coupling_for_link', &
         'Couplings can only be requested for variables that you registered yourself, &
         &not inherited ones such as the current ' // trim(link%name) // '.')

      ! Create a coupling task (reuse existing one if available, and not user-specified)
      call self%coupling_task_list%add(link, .false., task)
   end subroutine create_coupling_task

   subroutine request_coupling_for_link(self, link, master)
      class (type_base_model),  intent(inout) :: self
      type (type_link), target, intent(inout) :: link
      character(len=*),         intent(in)    :: master

      class (type_coupling_task), pointer :: task

      ! Create a coupling task (reuse existing one if available, and not user-specified)
      call create_coupling_task(self, link, task)
      if (.not. associated(task)) return   ! We already have a user-specified task, which takes priority

      ! Configure coupling task
      task%master_name = master
   end subroutine request_coupling_for_link

   recursive subroutine request_coupling_for_name(self, slave, master)
      class (type_base_model), intent(inout), target :: self
      character(len=*),        intent(in)            :: slave, master

      class (type_base_model), pointer :: parent
      type (type_link),        pointer :: link
      integer                          :: islash

      ! If a path with / is given, redirect to tentative parent model.
      islash = index(slave, '/', .true.)
      if (islash /= 0) then
         parent => self%find_model(slave(:islash - 1))
         call request_coupling_for_name(parent, slave(islash + 1:), master)
         return
      end if

      link => self%links%find(slave)
      if (.not. associated(link)) call self%fatal_error('request_coupling_for_name', &
         'Specified slave (' // trim(slave) // ') not found. Make sure the variable is registered before calling requ
est_coupling.')
      call request_coupling_for_link(self, link, master)
   end subroutine request_coupling_for_name

   subroutine request_coupling_for_id(self, id, master)
      class (type_base_model),  intent(inout) :: self
      class (type_variable_id), intent(inout) :: id
      character(len=*),         intent(in)    :: master

      if (.not. associated(id%link)) call self%fatal_error('request_coupling_for_id', &
         'The provided variable identifier has not been registered yet.')
      call self%request_coupling(id%link, master)
   end subroutine request_coupling_for_id

   subroutine request_standard_coupling_for_link(self, link, master)
      use fabm_standard_variables   ! workaround for bug in Cray compiler 8.3.4
      class (type_base_model),                         intent(inout)      :: self
      type (type_link), target,                        intent(inout)      :: link
      class (type_domain_specific_standard_variable), intent(in), target :: master

      class (type_coupling_task), pointer :: task

      call create_coupling_task(self, link, task)
      if (.not. associated(task)) return   ! We already have a user-specified task, which takes priority
      task%master_standard_variable => master%typed_resolve()
   end subroutine request_standard_coupling_for_link

   subroutine request_standard_coupling_for_id(self, id, master)
      class (type_base_model),                         intent(inout)      :: self
      class (type_variable_id),                        intent(inout)      :: id
      class (type_domain_specific_standard_variable), intent(in), target :: master

      if (.not. associated(id%link)) call self%fatal_error('request_standard_coupling_for_id', &
         'The provided variable identifier has not been registered yet.')
      call self%request_standard_coupling_for_link(id%link, master)
   end subroutine request_standard_coupling_for_id

   subroutine integer_pointer_set_append(self, value)
      class (type_integer_pointer_set), intent(inout) :: self
      integer, target                                 :: value

      type (type_integer_pointer), allocatable :: oldarray(:)

      ! Create a new list of integer 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 integer to the list.
      self%pointers(size(self%pointers))%p => value
      self%pointers(size(self%pointers))%p = self%value
   end subroutine integer_pointer_set_append

   subroutine integer_pointer_set_extend(self, other)
      class (type_integer_pointer_set), intent(inout) :: self
      class (type_integer_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 integer_pointer_set_extend

   subroutine integer_pointer_set_finalize(self)
      class (type_integer_pointer_set), intent(inout) :: self

      if (allocated(self%pointers)) deallocate(self%pointers)
      self%value = -1
   end subroutine integer_pointer_set_finalize

   subroutine integer_pointer_set_set_value(self, value)
      class (type_integer_pointer_set), intent(inout) :: self
      integer,                          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
      self%value = value
   end subroutine integer_pointer_set_set_value

   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)
```

```fortran
      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

      source_ = source_do_bottom
      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_bottom, 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_bottom_source

   subroutine register_bottom_state_variable(self, id, name, units, long_name, &
                                             initial_value, minimum, maximum, missing_value, &
                                             standard_variable, presence, background_value)
      class (type_base_model),              intent(inout)         :: self
      type (type_bottom_state_variable_id), intent(inout), target :: id
      character(len=*),                     intent(in)            :: name, long_name, units
      real(rk),                             intent(in), optional  :: initial_value
      real(rk),                             intent(in), optional  :: minimum, maximum, missing_value, background_valu
e
      class (type_base_standard_variable),  intent(in), optional  :: standard_variable
      integer,                              intent(in), optional  :: presence

      call self%add_horizontal_variable(name, units, long_name, missing_value, minimum, maximum, &
                                        initial_value=initial_value, background_value=background_value, &
                                        standard_variable=standard_variable, presence=presence, domain=domain_bottom,
 &
                                        state_index=id%bottom_state_index, read_index=id%horizontal_index, &
                                        background=id%background, link=id%link, source=source_state)
      call register_bottom_source(self, id%link, id%bottom_sms)
   end subroutine register_bottom_state_variable

   subroutine register_surface_state_variable(self, id, name, units, long_name, &
                                              initial_value, minimum, maximum, missing_value, &
                                              standard_variable, presence, background_value)
      class (type_base_model),               intent(inout)         :: self
      type (type_surface_state_variable_id), intent(inout), target :: id
      character(len=*),                      intent(in)            :: name, long_name, units
      real(rk),                              intent(in), optional  :: initial_value
      real(rk),                              intent(in), optional  :: minimum, maximum, missing_value, background_val
ue
      class (type_base_standard_variable),   intent(in), optional  :: standard_variable
      integer,                               intent(in), optional  :: presence

      call self%add_horizontal_variable(name, units, long_name, missing_value, minimum, maximum, &
                                        initial_value=initial_value, background_value=background_value, &
                                        standard_variable=standard_variable, presence=presence, domain=domain_surface
, &
                                        state_index=id%surface_state_index, read_index=id%horizontal_index, &
                                        background=id%background, link=id%link, source=source_state)
      call register_surface_source(self, id%link, id%surface_sms)
   end subroutine register_surface_state_variable

   subroutine add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
                           initial_value, background_value, fill_value, standard_variable, presence, output, source,
&
                           act_as_state_variable, read_index, state_index, write_index, background, link)
      class (type_base_model),       target,intent(inout)       :: self
      type (type_internal_variable),pointer                     :: variable
      character(len=*),              target,intent(in)          :: name
      character(len=*),                     intent(in), optional :: long_name, units
      real(rk),                             intent(in), optional :: minimum, maximum, missing_value
      real(rk),                             intent(in), optional :: initial_value, background_value, fill_value
      class (type_base_standard_variable),  intent(in), optional :: standard_variable
      integer,                              intent(in), optional :: presence, output, source
      logical,                              intent(in), optional :: act_as_state_variable
      integer,                       target,            optional :: read_index, state_index, write_index
      real(rk),                      target,            optional :: background
      type (type_link),              pointer,           optional :: link

      integer                   :: length, i
      character(len=256)        :: text
      type (type_link), pointer :: link_
      class (type_base_standard_variable), pointer :: pstandard_variable

      ! Check whether the model information may be written to (only during initialization)
      if (self%frozen) call self%fatal_error('add_variable', &
         'Cannot register variable "' // trim(name) // '" because the model initialization phase has already complete
d &
         &(initialize has been called).')

      ! Ascertain whether the provided name is valid.
      length = len_trim(name)
      if (length > len(variable%name)) then
         call self%fatal_error('add_variable', 'Variable name "' // trim(name) // '" exceeds maximum length.')
      elseif (length == 0) then
         call self%fatal_error('add_variable', 'Cannot register variable with empty name "".')
      elseif (name(length:length) == '*') then
         ! Last character is an asterisk (*) that needs to be replaced with an integer than makes the name unique.
         i = 1
         do
            write (variable%name,'(A,I0)') name(:length - 1), i
            if (.not. associated(self%links%find(variable%name))) exit
            i = i + 1
         end do
      elseif (name /= get_safe_name(name)) then
```

```
         call self%fatal_error('add_variable', 'Cannot register variable "' // trim(name) // '" because its name is n
ot valid. &
            &Variable names can contain letters, digits and underscores only.')
      else
         variable%name = name
      end if

      if (present(write_index) .and. .not. present(source)) call self%fatal_error('add_variable', &
         'Cannot register writable variable "' // trim(name) // '" because "source" argument is not provided.')

      variable%owner => self
      if (present(units)) variable%units = units
      if (present(long_name)) then
         variable%long_name = long_name
      else
         variable%long_name = variable%name
      end if
      if (present(minimum))       variable%minimum       = minimum
      if (present(maximum))       variable%maximum       = maximum
      if (present(missing_value)) variable%missing_value = missing_value
      if (present(initial_value)) variable%initial_value = initial_value
      if (present(presence))      variable%presence      = presence
      if (present(act_as_state_variable)) variable%fake_state_variable = act_as_state_variable
      if (present(output))        variable%output        = output
      if (present(source))        variable%source        = source
      variable%prefill_value = variable%missing_value
      if (present(fill_value)) then
         variable%prefill = prefill_constant
         variable%prefill_value = fill_value
      end if
      if (present(standard_variable)) then
         pstandard_variable => standard_variable%resolve()
         select type (pstandard_variable)
         class is (type_domain_specific_standard_variable)
            call variable%standard_variables%add(pstandard_variable)
         class is (type_universal_standard_variable)
            select case (variable%domain)
            case (domain_interior);   call variable%standard_variables%add(pstandard_variable%in_interior())
            case (domain_surface);    call variable%standard_variables%add(pstandard_variable%at_surface())
            case (domain_bottom);     call variable%standard_variables%add(pstandard_variable%at_bottom())
            case (domain_horizontal); call variable%standard_variables%add(pstandard_variable%at_interfaces())
            end select
         end select
      end if

      if (present(state_index)) then
         ! Ensure that initial value falls within prescribed valid range.
         if (variable%initial_value < variable%minimum .or. variable%initial_value > variable%maximum) then
            write (text,*) 'Initial value', variable%initial_value, 'for variable "' // trim(name) // '" lies&
                  &outside allowed range', variable%minimum, 'to', variable%maximum
            call self%fatal_error('fill_internal_variable', text)
         end if

         ! Store a pointer to the variable that should hold the state variable index.
         call variable%state_indices%append(state_index)
      end if

      if (present(background)) then
         ! Store a pointer to the variable that should hold the background value.
         ! If the background value itself is also prescribed, use it.
         call variable%background_values%append(background)
         if (present(background_value)) call variable%background_values%set_value(background_value)
      end if

      if (present(read_index)) then
         variable%read_index => read_index
         call variable%read_indices%append(read_index)
      end if
      if (present(write_index)) then
         variable%write_index => write_index
         _ASSERT_(variable%source /= source_state, 'add_variable', 'Variable ' // trim(name) // ' being registered wi
th source_state and write index.')
         call variable%write_indices%append(write_index)
      end if

      ! Create a class pointer and use that to create a link.
      link_ => add_object(self, variable)
      if (present(link)) then
         if (associated(link)) call self%fatal_error('add_variable', 'Identifier supplied for ' // trim(name) // ' is
 already associated with ' // trim(link%name) // '.')
         link => link_
      end if
   end subroutine add_variable

   subroutine add_interior_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &
                                    background_value, fill_value, specific_light_extinction, &
                                    no_precipitation_dilution, no_river_dilution, standard_variable, presence,
output, &
                                    act_as_state_variable, source, &
                                    read_index, state_index, write_index, &
                                    background, link)
      class (type_base_model),target,      intent(inout)        :: self
      character(len=*),                    intent(in)           :: name
      character(len=*),                    intent(in), optional :: units, long_name
      real(rk),                            intent(in), optional :: minimum, maximum, missing_value
      real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
      real(rk),                            intent(in), optional :: specific_light_extinction
      logical,                             intent(in), optional :: no_precipitation_dilution, no_river_dilution
```

```fortran
      class (type_base_standard_variable), intent(in), optional :: standard_variable
      integer,                             intent(in), optional :: presence, output, source
      logical,                             intent(in), optional :: act_as_state_variable
      integer,  target,                                optional :: read_index, state_index, write_index
      real(rk), target,                                optional :: background
      type (type_link), pointer,                       optional :: link

      type (type_internal_variable), pointer :: variable

      allocate(variable)
      variable%domain = domain_interior

      ! Fill fields specific to interior variables.
      if (present(no_precipitation_dilution)) variable%no_precipitation_dilution = no_precipitation_dilution
      if (present(no_river_dilution))         variable%no_river_dilution         = no_river_dilution
      if (present(specific_light_extinction)) call variable%contributions%add( &
         standard_variables%attenuation_coefficient_of_photosynthetic_radiative_flux, scale_factor=specific_light_extinction)

      ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
      call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
                        initial_value, background_value, fill_value, standard_variable, presence, output, source, &
                        act_as_state_variable, read_index, state_index, write_index, background, link)
   end subroutine add_interior_variable

   subroutine add_horizontal_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &
                                                background_value, fill_value, standard_variable, presence, output, &
                                                act_as_state_variable, domain, source, &
                                                read_index, state_index, write_index, background, link)
      class (type_base_model),target,      intent(inout)        :: self
      character(len=*),                    intent(in)           :: name
      character(len=*),                    intent(in), optional :: units, long_name
      real(rk),                            intent(in), optional :: minimum, maximum, missing_value
      real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
      class (type_base_standard_variable), intent(in), optional :: standard_variable
      integer,                             intent(in), optional :: presence, domain, output, source
      logical,                             intent(in), optional :: act_as_state_variable
      integer,  target,                                optional :: read_index, state_index, write_index
      real(rk), target,                                optional :: background
      type (type_link), pointer,                       optional :: link

      type (type_internal_variable), pointer :: variable

      allocate(variable)
      variable%domain = domain_horizontal
      if (present(domain)) variable%domain = domain

      ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
      call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
                        initial_value, background_value, fill_value, standard_variable, presence, output, source, &
                        act_as_state_variable, read_index, state_index, write_index, background, link)
   end subroutine add_horizontal_variable

   subroutine add_scalar_variable(self, name, units, long_name, missing_value, minimum, maximum, initial_value, &
                                  background_value, fill_value, standard_variable, presence, output, &
                                  read_index, state_index, write_index, sms_index, background, link)
      class (type_base_model),target,      intent(inout)        :: self
      character(len=*),                    intent(in)           :: name
      character(len=*),                    intent(in), optional :: units, long_name
      real(rk),                            intent(in), optional :: minimum, maximum, missing_value
      real(rk),                            intent(in), optional :: initial_value, background_value, fill_value
      class (type_base_standard_variable), intent(in), optional :: standard_variable
      integer,                             intent(in), optional :: presence, output
      integer,  target,                                optional :: read_index, state_index, write_index, sms_index
      real(rk), target,                                optional :: background
      type (type_link), pointer,                       optional :: link

      type (type_internal_variable), pointer :: variable

      allocate(variable)
      variable%domain = domain_scalar

      ! Process remainder of fields and creation of link generically (i.e., irrespective of variable domain).
      call add_variable(self, variable, name, units, long_name, missing_value, minimum, maximum, &
                        initial_value, background_value, fill_value, standard_variable, presence, output, source_unknown, &
                        .false., read_index, state_index, write_index, background, link)
   end subroutine add_scalar_variable

   recursive function add_object(self, object) result(link)
      ! This subroutine creates a link to the supplied object, then allows
      ! parent models to do the same.
      ! NB this subroutine MUST be recursive, to allow parent models to override
      ! the properties of objects added by their child models.
      class (type_base_model), target, intent(inout) :: self
      type (type_internal_variable), pointer         :: object

      type (type_link), pointer         :: link, parent_link
      character(len=attribute_length)   :: oriname
      integer                           :: instance
      logical                           :: duplicate
      type (type_link_pointer), pointer :: link_pointer

      ! First check if a link with this name exists.
      duplicate = associated(self%links%find(object%name))

      if (duplicate) then
```

```
         ! Link with this name exists already.
         ! Append numbers to the variable name until a unique name is found.
         oriname = object%name
         instance = 0
         do
            write (object%name,'(a,a,i0)') trim(oriname), '_', instance
            if (.not. associated(self%links%find(object%name))) exit
            instance = instance + 1
         end do
      end if

      ! Create link for this object.
      link => self%links%append(object, object%name)

      ! Store a pointer to the link with the object to facilitate redirection of the link during coupling.
      allocate(link_pointer)
      link_pointer%p => link
      link_pointer%next => object%first_link
      object%first_link => link_pointer

      ! If this name matched that of a previous variable, create a coupling to it.
      if (duplicate) call self%request_coupling(link, oriname)

      ! Forward to parent
      if (associated(self%parent)) then
         if (len_trim(self%name) + 1 + len_trim(object%name) > len(object%name)) call self%fatal_error('add_object', 
&
            'Variable path "' // trim(self%name) // '/' // trim(object%name) // '" exceeds maximum allowed length.')
         object%name = trim(self%name) // '/' // trim(object%name)

         ! Below, the equivalent self%parent%add_object(object) confuses PGI 18.10 (Jorn 2019-04-24)
         parent_link => add_object(self%parent, object)
      end if
   end function add_object

   subroutine register_interior_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variabl
e, output, &
                                                    source, act_as_state_variable, prefill_value)
      class (type_base_model),            intent(inout), target :: self
      type (type_diagnostic_variable_id),  intent(inout), target :: id
      character(len=*),                   intent(in)           :: name, long_name, units
      integer,                            intent(in), optional :: output, source
      real(rk),                           intent(in), optional :: missing_value, prefill_value
      class (type_base_standard_variable), intent(in), optional :: standard_variable
      logical,                            intent(in), optional :: act_as_state_variable

      integer :: source_

      source_ = source_do
      if (present(source)) source_ = source
      call self%add_interior_variable(name, units, long_name, missing_value, fill_value=prefill_value, &
         standard_variable=standard_variable, output=output, source=source_, write_index=id%write_index, link=id%link
, &
         act_as_state_variable=act_as_state_variable)
   end subroutine register_interior_diagnostic_variable

   subroutine register_horizontal_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_varia
ble, output, &
                                                      source, act_as_state_variable, domain)
      class (type_base_model),                       intent(inout), target :: self
      type (type_horizontal_diagnostic_variable_id), intent(inout), target :: id
      character(len=*),                              intent(in)            :: name, units, long_name
      integer,                                       intent(in)            :: source
      integer,                                       intent(in), optional  :: output, domain
      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

      call self%add_horizontal_variable(name, units, long_name, missing_value, &
                                        standard_variable=standard_variable, output=output, &
                                        source=source, write_index=id%horizontal_write_index, link=id%link, &
                                        act_as_state_variable=act_as_state_variable, domain=domain)
   end subroutine register_horizontal_diagnostic_variable

   subroutine register_surface_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variable
, &
                                                   output, source, act_as_state_variable)
      class (type_base_model),                    intent(inout), target :: self
      type (type_surface_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_surface
      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%surface_write_index, link=id%link, &
                                        act_as_state_variable=act_as_state_variable, domain=domain_surface)
   end subroutine register_surface_diagnostic_variable

   subroutine register_bottom_diagnostic_variable(self, id, name, units, long_name, missing_value, standard_variable,
 &
                                                  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
```

```fortran

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

      call register_surface_state_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_state_dependency

   subroutine register_standard_surface_state_dependency2(self, id, standard_variable, required)
      class (type_base_model),                       intent(inout) :: self
      type (type_surface_state_variable_id), target, intent(inout) :: id
      type (type_horizontal_standard_variable),      intent(in)    :: standard_variable
      logical, optional,                             intent(in)    :: required

      call register_surface_state_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_state_dependency2

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

      call register_named_interior_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_interior_dependency

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

      call register_standard_interior_dependency(self, id, standard_variable%in_interior(), required)
   end subroutine register_universal_interior_dependency

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

      call register_named_horizontal_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_horizontal_dependency

   subroutine register_standard_horizontal_dependency2(self, id, standard_variable, required)
      class (type_base_model),               intent(inout)         :: self
      type (type_horizontal_dependency_id),  intent(inout), target :: id
      type (type_surface_standard_variable), intent(in)            :: standard_variable
      logical, optional,                     intent(in)            :: required

      call register_named_horizontal_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_horizontal_dependency2

   subroutine register_standard_horizontal_dependency3(self, id, standard_variable, required)
      class (type_base_model),              intent(inout)         :: self
      type (type_horizontal_dependency_id), intent(inout), target :: id
      type (type_bottom_standard_variable), intent(in)            :: standard_variable
      logical, optional,                    intent(in)            :: required

      call register_named_horizontal_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_horizontal_dependency3

   subroutine register_universal_horizontal_dependency(self, id, standard_variable, domain, required)
      class (type_base_model),                 intent(inout)         :: self
      type (type_horizontal_dependency_id),    intent(inout), target :: id
      type (type_universal_standard_variable), intent(in)            :: standard_variable
      integer, optional,                       intent(in)            :: domain
      logical, optional,                       intent(in)            :: required

      integer :: domain_

      domain_ = domain_horizontal
      if (present(domain)) domain_ = domain
      select case (domain_)
      case (domain_surface);    call register_standard_horizontal_dependency2(self, id, standard_variable%at_surface(), required)
      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
```

```fortran
      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

      class (type_horizontal_expression), allocatable :: copy

      allocate(copy, source=expression)
      copy%out => id%horizontal_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

   recursive subroutine register_expression(self, expression)
      class (type_base_model), intent(inout) :: self
      class (type_expression), intent(in)    :: expression

      class (type_expression), pointer :: current

      if (.not. associated(self%first_expression)) then
         allocate(self%first_expression, source=expression)
         current => self%first_expression
      else
         current => self%first_expression
         do while (associated(current%next))
            current => current%next
         end do
         allocate(current%next, source=expression)
         current => current%next
      end if

      if (associated(self%parent)) call register_expression(self%parent, expression)
   end subroutine

   subroutine get_real_parameter(self, value, name, units, long_name, default, scale_factor, minimum, maximum)
      class (type_base_model), intent(inout), target  :: self
      real(rk),                intent(inout)          :: value
      character(len=*),        intent(in)             :: name
      character(len=*),        intent(in),   optional :: units, long_name
      real(rk),                intent(in),   optional :: default, scale_factor, minimum, maximum

      class (type_property), pointer :: property
      logical                        :: success
      type (type_real_property)      :: current_parameter
      character(len=13)              :: text1, text2

      if (present(default)) then
         current_parameter%has_default = .true.
         current_parameter%default = default
         value = default
      end if

      ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
      property => self%parameters%find_in_tree(name)
      if (associated(property)) then
         ! Value found - try to convert to real.
         value = property%to_real(success=success)
         if (.not. success) call self%fatal_error('get_real_parameter', &
            'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not a real number.'
)
      elseif (.not.present(default)) then
         call self%fatal_error('get_real_parameter', 'No value provided for parameter "' // trim(name) // '".')
      end if

      if (present(minimum)) then
         if (value < minimum) then
            write (text1,'(G13.6)') value
            write (text2,'(G13.6)') minimum
            call self%fatal_error('get_real_parameter', 'Value ' // trim(adjustl(text1)) // ' for parameter "' // tri
m(name) &
               // '" is less than prescribed minimum of ' // trim(adjustl(text2)) // '.')
         end if
      end if
      if (present(maximum)) then
         if (value > maximum) then
            write (text1,'(G13.6)') value
            write (text2,'(G13.6)') maximum
            call self%fatal_error('get_real_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // trim
(name) &
               // '" exceeds prescribed maximum of ' // trim(adjustl(text2)) // '.')
         end if
      end if

      ! Store parameter settings
      current_parameter%value = value
      call set_parameter(self, current_parameter, name, units, long_name)

      ! Apply scale factor to value provided to the model (if requested).
      if (present(scale_factor)) value = value * scale_factor
   end subroutine get_real_parameter

   subroutine set_parameter(self, parameter, name, units, long_name)
      class (type_base_model), intent(inout), target :: self
      class (type_property),   intent(inout)         :: parameter
      character(len=*),        intent(in)            :: name
      character(len=*),        intent(in), optional  :: units, long_name

      parameter%name = name
      if (present(units))     parameter%units     = units
      if (present(long_name)) parameter%long_name = long_name
      call self%parameters%set_in_tree(parameter)
   end subroutine set_parameter

   subroutine get_integer_parameter(self, value, name, units, long_name, default, minimum, maximum)
      class (type_base_model), intent(inout), target :: self
      integer,                 intent(inout)         :: value
      character(len=*),        intent(in)            :: name
      character(len=*),        intent(in), optional  :: units, long_name
      integer,                 intent(in), optional  :: default, minimum, maximum

      class (type_property), pointer :: property
      type (type_integer_property)   :: current_parameter
      logical                        :: success
      character(len=8)               :: text1, text2

      if (present(default)) then
         current_parameter%has_default = .true.
         current_parameter%default = default
         value = default
      end if

      ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
      property => self%parameters%find_in_tree(name)
      if (associated(property)) then
         ! Value found - try to convert to integer.
         value = property%to_integer(success=success)
         if (.not. success) call self%fatal_error('get_integer_parameter', &
            'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not an integer numb
er.')
      elseif (.not.present(default)) then
         call self%fatal_error('get_integer_parameter', 'No value provided for parameter "' // trim(name) // '".')
      end if

      if (present(minimum)) then
         if (value < minimum) then
            write (text1,'(I0)') value
            write (text2,'(I0)') minimum
            call self%fatal_error('get_integer_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // t
rim(name) &
               // '" is less than prescribed minimum of ' // trim(adjustl(text2)) // '.')
         end if
      end if
      if (present(maximum)) then
         if (value > maximum) then
            write (text1,'(I0)') value
            write (text2,'(I0)') maximum
            call self%fatal_error('get_integer_parameter','Value ' // trim(adjustl(text1)) // ' for parameter "' // t
rim(name) &
               //'" exceeds prescribed maximum of ' // trim(adjustl(text2)) // '.')
         end if
      end if

      ! Store parameter settings
      current_parameter%value = value
      call set_parameter(self, current_parameter, name, units, long_name)
   end subroutine get_integer_parameter

   subroutine get_logical_parameter(self, value, name, units, long_name, default)
      class (type_base_model), intent(inout), target :: self
      logical,                 intent(inout)         :: value
      character(len=*),        intent(in)            :: name
      character(len=*),        intent(in), optional  :: units, long_name
      logical,                 intent(in), optional  :: default

      class (type_property), pointer :: property
      type (type_logical_property)   :: current_parameter
      logical                        :: success

      if (present(default)) then
         current_parameter%has_default = .true.
         current_parameter%default = default
         value = default
      end if

      ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
      property => self%parameters%find_in_tree(name)
      if (associated(property)) then
         ! Value found - try to convert to logical.
         value = property%to_logical(success=success)
         if (.not. success) call self%fatal_error('get_logical_parameter', &
            'Value "' // trim(property%to_string()) // '" for parameter "' // trim(name) // '" is not a Boolean value
.')
      elseif (.not. present(default)) then
         call self%fatal_error('get_logical_parameter', 'No value provided for parameter "' // trim(name) // '".')
      end if

      ! Store parameter settings
      current_parameter%value = value
      call set_parameter(self, current_parameter, name, units, long_name)
   end subroutine get_logical_parameter

   recursive subroutine get_string_parameter(self, value, name, units, long_name, default)
      class (type_base_model), intent(inout), target :: self
      character(len=*),        intent(inout)         :: value
      character(len=*),        intent(in)            :: name
      character(len=*),        intent(in), optional  :: units, long_name
      character(len=*),        intent(in), optional  :: default

      class (type_property), pointer :: property
      type (type_string_property)    :: current_parameter
      logical                        :: success

      if (present(default)) then
         current_parameter%has_default = .true.
         current_parameter%default = default
         value = default
      end if

      ! Try to find a user-specified value for this parameter in our dictionary, and in those of our ancestors.
      property => self%parameters%find_in_tree(name)
      if (associated(property)) then
         ! Value found - try to convert to string.
         value = property%to_string(success=success)
         if (.not. success) call self%fatal_error('get_string_parameter', &
            'Value for parameter "' // trim(name) // '" cannot be converted to string.')
      elseif (.not. present(default)) then
         call self%fatal_error('get_string_parameter','No value provided for parameter "' // trim(name) // '".')
      end if

      ! Store parameter settings
      current_parameter%value = value
      call set_parameter(self, current_parameter, name, units, long_name)
   end subroutine get_string_parameter

   function find_object(self, name, recursive, exact) result(object)
      class (type_base_model),  intent(in), target :: self
      character(len=*),         intent(in)         :: name
      logical,        optional, intent(in)         :: recursive, exact
      type (type_internal_variable), pointer       :: object

      type (type_link), pointer :: link

      object => null()
      link => self%find_link(name, recursive, exact)
      if (associated(link)) object => link%target

   end function find_object

   recursive function find_link(self, name, recursive, exact) result(link)
      class (type_base_model),  intent(in), target :: self
      character(len=*),         intent(in)         :: name
      logical,        optional, intent(in)         :: recursive, exact
      type (type_link), pointer                    :: link

      integer                         :: n
      logical                         :: recursive_eff, exact_eff
      class (type_base_model),pointer :: current

      link => null()

      n = len_trim(name)
      if (n >= 1) then
         if (name(1:1) == '/') then
            link => find_link(self, name(2:), recursive, exact=.true.)
            return
         end if
         if (n >= 2) then
            if (name(1:2) == './') then
               link => find_link(self, name(3:), recursive, exact=.true.)
               return
            end if
            if (n >= 3) then
               if (name(1:3) == '../') then
                  if (.not. associated(self%parent)) return
                  link => find_link(self%parent, name(4:), recursive, exact=.true.)
                  return
               end if
            end if
         end if
      end if

      recursive_eff = .false.
      if (present(recursive)) recursive_eff = recursive

      ! First search self and ancestors (if allowed) based on exact name provided.
      current => self
      do while (associated(current))
         link => current%links%find(name)
         if (associated(link)) return
         if (.not. recursive_eff) exit
         current => current%parent
      end do

      exact_eff = .true.
      if (present(exact)) exact_eff = exact
      if (exact_eff) return

      ! Not found. Now search self and ancestors (if allowed) based on safe name (letters and underscores only).
      current => self
      do while (associated(current))
         link => current%links%first
         do while (associated(link))
            if (get_safe_name(link%name) == name) return
            link => link%next
         end do
```

```fortran
         if (.not. recursive_eff) exit
         current => current%parent
      end do
   end function find_link

   function find_model(self, name, recursive) result(found_model)
      class (type_base_model), target, intent(in) :: self
      character(len=*),                intent(in) :: name
      logical, optional,               intent(in) :: recursive
      class (type_base_model), pointer            :: found_model

      class (type_base_model), pointer     :: current_root
      logical                              :: recursive_eff
      type (type_model_list_node), pointer :: node
      integer                              :: istart, length

      found_model => null()

      ! Determine whether to also try among ancestors
      recursive_eff = .false.
      if (present(recursive)) recursive_eff = recursive .and. .not. (name == '.' .or. name == '..' &
           .or. name(:min(2, len(name))) == './' .or. name(:min(3, len(name))) == '../')

      current_root => self
      do while (associated(current_root))
         ! Process individual path components (separated by /)
         found_model => current_root
         istart = 1
         do while (associated(found_model) .and. istart <= len(name))
            length = index(name(istart:), '/') - 1
            if (length == -1) length = len(name) - istart + 1
            if (length == 2 .and. name(istart:istart + length - 1) == '..') then
               found_model => found_model%parent
            elseif (.not. (length == 1 .and. name(istart:istart + length - 1) == '.')) then
               node => found_model%children%find(name(istart:istart + length - 1))
               found_model => null()
               if (associated(node)) found_model => node%model
            end if
            istart = istart + length + 1
         end do

         ! Only continue if we have not found the model and are allowed to try parent model.
         if (associated(found_model) .or. .not. recursive_eff) return

         current_root => current_root%parent
      end do
   end function find_model

   function get_aggregate_variable_access(self, standard_variable) result(aggregate_variable_access)
      class (type_base_model),                         intent(inout) :: self
      class (type_domain_specific_standard_variable), target        :: standard_variable

      type (type_aggregate_variable_access), pointer :: aggregate_variable_access
      logical,                               pointer :: pmember

      ! First try to locate existing requests object for the specified standard variable.
      aggregate_variable_access => self%first_aggregate_variable_access
      pmember => standard_variable%aggregate_variable
      do while (associated(aggregate_variable_access))
         ! Note: for Cray 10.0.4, the comparison below fails for class pointers! Therefore we compare type member ref
erences.
         if (associated(pmember, aggregate_variable_access%standard_variable%aggregate_variable)) return
         aggregate_variable_access => aggregate_variable_access%next
      end do

      ! Not found - create a new requests object.
      allocate(aggregate_variable_access)
      aggregate_variable_access%standard_variable => standard_variable
      aggregate_variable_access%next => self%first_aggregate_variable_access
      self%first_aggregate_variable_access => aggregate_variable_access
   end function get_aggregate_variable_access

   function get_free_unit() result(unit)
      integer :: unit
      integer, parameter :: LUN_MIN=10, LUN_MAX=1000

      logical :: opened

      do unit = LUN_MIN, LUN_MAX
         inquire(unit=unit, opened=opened)
         if (.not. opened) return
      end do
      unit = -1
   end function get_free_unit

   function get_safe_name(name) result(safe_name)
      character(len=*), intent(in) :: name
      character(len=len(name))     :: safe_name

      integer :: i, ch
      logical :: valid

      safe_name = name
      do i = 1, len_trim(name)
         ch = iachar(name(i:i))
         valid = (ch >= iachar('a') .and. ch <= iachar('z')) & ! Lower-case letter
            .or. (ch >= iachar('A') .and. ch <= iachar('Z')) & ! Upper-case letter
            .or. (ch >= iachar('0') .and. ch <= iachar('9')) & ! Number
            .or. (ch == iachar('_'))                             ! Underscore
         if (.not. valid) safe_name(i:i) = '_'
      end do
   end function

   recursive subroutine abstract_model_factory_initialize(self)
      class (type_base_model_factory), intent(inout) :: self

      type (type_base_model_factory_node), pointer :: current

      self%initialized = .true.
      current => self%first_child
      do while(associated(current))
         if (.not. current%factory%initialized) call current%factory%initialize()
         current => current%next
      end do
   end subroutine abstract_model_factory_initialize

   subroutine abstract_model_factory_add(self, child, prefix)
      class (type_base_model_factory),         intent(inout) :: self
      class (type_base_model_factory), target, intent(in)    :: child
      character(len=*), optional,              intent(in)    :: prefix

      type (type_base_model_factory_node), pointer :: current

      if (self%initialized) call driver%fatal_error('abstract_model_factory_add', &
         'BUG! Factory initialiation is complete. Child factories can no longer be added.')

      if (.not.associated(self%first_child)) then
         allocate(self%first_child)
         current => self%first_child
      else
         current => self%first_child
         do while(associated(current%next))
            current => current%next
         end do
         allocate(current%next)
         current => current%next
      end if

      current%factory => child
      if (present(prefix)) current%prefix = prefix
   end subroutine abstract_model_factory_add

   recursive subroutine abstract_model_factory_create(self, name, model)
      class (type_base_model_factory), intent(in) :: self
      character(len=*),                intent(in) :: name
      class (type_base_model), pointer            :: model

      type (type_base_model_factory_node), pointer :: child
      integer                                      :: n

      child => self%first_child
      do while(associated(child))
         if (child%prefix /= '') then
            n = len_trim(child%prefix)
            if (len_trim(name) > n + 1) then
               if (name(1:n) == child%prefix .and. (name(n + 1:n + 1) == '_' .or. name(n + 1:n + 1) == '/')) &
                  call child%factory%create(name(n+2:), model)
            end if
         else
            call child%factory%create(name, model)
         end if
         if (associated(model)) return
         child => child%next
      end do
   end subroutine abstract_model_factory_create

   recursive subroutine abstract_model_factory_register_version(self, name, version_string)
      class (type_base_model_factory),intent(in) :: self
      character(len=*),               intent(in) :: name, version_string

      type (type_version), pointer :: version

      if (associated(first_module_version)) then
         version => first_module_version
         do while (associated(version%next))
            version => version%next
         end do
         allocate(version%next)
         version => version%next
      else
         allocate(first_module_version)
         version => first_module_version
      end if
      version%module_name = name
      version%version_string = version_string
   end subroutine abstract_model_factory_register_version

   recursive subroutine abstract_model_factory_finalize(self)
      class (type_base_model_factory), intent(inout) :: self

      type (type_base_model_factory_node), pointer :: current, next

      current => self%first_child
      do while(associated(current))
         next => current%next
         call current%factory%finalize()
         deallocate(current)
         current => next
      end do
      self%first_child => null()
   end subroutine abstract_model_factory_finalize

   subroutine coupling_task_list_remove(self, task)
      class (type_coupling_task_list), intent(inout) :: self
      class (type_coupling_task), pointer            :: task
      if (associated(task%previous)) then
         task%previous%next => task%next
      else
         self%first => task%next
      end if
      if (associated(task%next)) task%next%previous => task%previous
      deallocate(task)
   end subroutine

   function coupling_task_list_add_object(self, task, always_create) result(used)
      class (type_coupling_task_list), intent(inout) :: self
      class (type_coupling_task), pointer            :: task
      logical,                         intent(in)    :: always_create
      logical                                        :: used

      class (type_coupling_task), pointer :: existing_task

      ! First try to find an existing coupling task for this link. If one exists, we'll replace it.
      used = .false.
      existing_task => self%first
      do while (associated(existing_task))
         ! Check if we have found an existing task for the same link.
         if (associated(existing_task%slave, task%slave)) then
            ! If existing one has higher priority, do not add the new task and return (used=.false.)
            if (existing_task%user_specified .and. .not. always_create) return

            ! We will overwrite the existing task - remove existing task and exit loop
            call self%remove(existing_task)
            exit
         end if
         existing_task => existing_task%next
      end do

      used = .true.
      if (.not. associated(self%first)) then
         ! Task list is empty - add first.
         self%first => task
         task%previous => null()
      else
         ! Task list contains items - append to tail.

         ! Find tail of the list
         existing_task => self%first
         do while (associated(existing_task%next))
            existing_task => existing_task%next
         end do

         existing_task%next => task
         task%previous => existing_task
      end if
      task%next => null()
   end function coupling_task_list_add_object

   subroutine coupling_task_list_add(self, link, always_create, task)
      class (type_coupling_task_list), intent(inout)         :: self
      type (type_link),                intent(inout), target :: link
      logical,                         intent(in)            :: always_create
      class (type_coupling_task), pointer                    :: task

      logical :: used

      allocate(task)
      task%slave => link
      used = self%add_object(task, always_create)
      if (.not. used) deallocate(task)
   end subroutine coupling_task_list_add

   character(len=32) function source2string(source)
      integer, intent(in) :: source
      select case (source)
      case (source_unknown);                  source2string = 'unknown'
      case (source_state);                    source2string = 'state'
      case (source_external);                 source2string = 'external'
      case (source_do);                       source2string = 'do'
      case (source_do_column);                source2string = 'do_column'
      case (source_do_horizontal);            source2string = 'do_horizontal'
      case (source_do_bottom);                source2string = 'do_bottom'
      case (source_do_surface);               source2string = 'do_surface'
      case (source_constant);                 source2string = 'constant'
      case (source_get_vertical_movement);    source2string = 'get_vertical_movement'
      case (source_check_state);              source2string = 'check_state'
      case (source_check_bottom_state);       source2string = 'check_bottom_state'
      case (source_check_surface_state);      source2string = 'check_surface_state'
      case (source_initialize_state);         source2string = 'initialize_state'
      case (source_initialize_bottom_state);  source2string = 'initialize_bottom_state'
      case (source_initialize_surface_state); source2string = 'initialize_surface_state'
      case (source_get_light_extinction);     source2string = 'get_light_extinction'
      case (source_get_drag);                 source2string = 'get_drag'
      case (source_get_albedo);               source2string = 'get_albedo'
```

```fortran
      case default
         write (source2string,'(i0)') source
      end select
   end function source2string

   subroutine variable_set_add(self, variable)
      class (type_variable_set), intent(inout) :: self
      type (type_internal_variable), target    :: variable

      type (type_variable_node), pointer :: node

      ! Check if this variable already exists.
      node => self%first
      do while (associated(node))
         if (associated(node%target, variable)) return
         node => node%next
      end do

      ! Create a new variable object and prepend it to the list.
      allocate(node)
      node%target => variable
      node%next => self%first
      self%first => node
   end subroutine variable_set_add

   subroutine variable_set_remove(self, variable, discard)
      class (type_variable_set), intent(inout) :: self
      type (type_internal_variable), target    :: variable
      logical, optional,         intent(in)    :: discard

      type (type_variable_node), pointer :: node, previous
      logical                            :: discard_

      ! Check if this variable already exists.
      previous => null()
      node => self%first
      do while (associated(node))
         if (associated(node%target, variable)) then
            if (associated(previous)) then
               previous%next => node%next
            else
               self%first => node%next
            end if
            deallocate(node)
            return
         end if
         previous => node
         node => node%next
      end do
      discard_ = .false.
      if (present(discard)) discard_ = discard
      if (.not. discard_) call driver%fatal_error('variable_set_remove', &
         'Variable "' // trim(variable%name) // '" not found in set.')
   end subroutine variable_set_remove

   logical function variable_set_contains(self, variable)
      class (type_variable_set), intent(in) :: self
      type (type_internal_variable), target :: variable

      type (type_variable_node), pointer :: node

      variable_set_contains = .true.
      node => self%first
      do while (associated(node))
         if (associated(node%target, variable)) return
         node => node%next
      end do
      variable_set_contains = .false.
   end function variable_set_contains

   subroutine variable_set_update(self, other)
      class (type_variable_set), intent(inout) :: self
      class (type_variable_set), intent(in)    :: other

      type (type_variable_node), pointer :: node

      node => other%first
      do while (associated(node))
         call self%add(node%target)
         node => node%next
      end do
   end subroutine variable_set_update

   subroutine variable_set_finalize(self)
      class (type_variable_set), 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()
   end subroutine variable_set_finalize

   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)
!-----------------------------------------------------------------------
```