```
! ========================================================
! Validate input symbols
! ========================================================

#ifndef _FABM_DIMENSION_COUNT_
#  error Preprocessor variable _FABM_DIMENSION_COUNT_ must be defined.
#endif

#if (_FABM_DIMENSION_COUNT_<0||_FABM_DIMENSION_COUNT_>3)
#  error Preprocessor variable _FABM_DIMENSION_COUNT_ takes values between 0 and 3 only.
#endif

#ifdef _FABM_DEPTH_DIMENSION_INDEX_
#  if (_FABM_DEPTH_DIMENSION_INDEX_<1)||(_FABM_DEPTH_DIMENSION_INDEX_>_FABM_DIMENSION_COUNT_)
#    error Preprocessor variable _FABM_DEPTH_DIMENSION_INDEX_ takes values between 1 and _FABM_DIMENSION_COUNT_ only.
#  endif
#endif

#ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
#  if (_FABM_VECTORIZED_DIMENSION_INDEX_<1)||(_FABM_VECTORIZED_DIMENSION_INDEX_>_FABM_DIMENSION_COUNT_)
#    error Preprocessor variable _FABM_VECTORIZED_DIMENSION_INDEX_ takes values between 1 and _FABM_DIMENSION_COUNT_ only.
#  endif
#endif

! ========================================================
! End of input symbol validation
! ========================================================

#ifndef _NO_DO_CONCURRENT_
#  define _DO_CONCURRENT_(iterator,start,stop) do concurrent (iterator=start:stop)
#  define _DO_CONCURRENT_WITH_STRIDE_(iterator,start,stop,stride) do concurrent (iterator=start:stop:stride)
#else
#  define _DO_CONCURRENT_(iterator,start,stop) do iterator=start,stop
#  define _DO_CONCURRENT_WITH_STRIDE_(iterator,start,stop,stride) do iterator=start,stop,stride
#endif

#if defined(_FABM_VECTORIZED_DIMENSION_INDEX_)||defined(_FABM_DEPTH_DIMENSION_INDEX_)
#  define _INTERIOR_IS_VECTORIZED_
#endif

#if defined(_FABM_VECTORIZED_DIMENSION_INDEX_)&&(_FABM_DEPTH_DIMENSION_INDEX_!=_FABM_VECTORIZED_DIMENSION_INDEX_)
#  define _HORIZONTAL_IS_VECTORIZED_
#endif

#define _I_ l__
#define _J_ m__
#define _N_ cache%n

#ifdef _INTERIOR_IS_VECTORIZED_
!  Interior fields are 1D
#  define _DIMENSION_SLICE_ ,dimension(:)
#  define _DIMENSION_SLICE_PLUS_1_ ,dimension(:,:)
#  define _DIMENSION_SLICE_PLUS_2_ ,dimension(:,:,:)
#  define _INDEX_SLICE_ (_I_)
#  define _INDEX_SLICE_PLUS_1_(i) (_I_,i)
#  define _INDEX_SLICE_PLUS_2_(i,j) (_I_,i,j)
#  define _DIMENSION_SLICE_AUTOMATIC_ ,dimension(_N_)
#else
!  Interior fields are 0D
#  define _DIMENSION_SLICE_
#  define _DIMENSION_SLICE_PLUS_1_ ,dimension(:)
#  define _DIMENSION_SLICE_PLUS_2_ ,dimension(:,:)
#  define _INDEX_SLICE_
#  define _INDEX_SLICE_PLUS_1_(i) (i)
#  define _INDEX_SLICE_PLUS_2_(i,j) (i,j)
#  define _DIMENSION_SLICE_AUTOMATIC_
#endif

#ifdef _HORIZONTAL_IS_VECTORIZED_
!  Horizontal fields are 1D
#  define _DIMENSION_HORIZONTAL_SLICE_ ,dimension(:)
#  define _DIMENSION_HORIZONTAL_SLICE_PLUS_1_ ,dimension(:,:)
#  define _DIMENSION_HORIZONTAL_SLICE_PLUS_2_ ,dimension(:,:,:)
#  define _INDEX_HORIZONTAL_SLICE_ (_J_)
#  define _INDEX_HORIZONTAL_SLICE_PLUS_1_(i) (_J_,i)
#  define _INDEX_HORIZONTAL_SLICE_PLUS_2_(i,j) (_J_,i,j)
#  define _DIMENSION_HORIZONTAL_SLICE_AUTOMATIC_ ,dimension(_N_)
#else
!  Horizontal fields are 0D
#  define _DIMENSION_HORIZONTAL_SLICE_
#  define _DIMENSION_HORIZONTAL_SLICE_PLUS_1_ ,dimension(:)
#  define _DIMENSION_HORIZONTAL_SLICE_PLUS_2_ ,dimension(:,:)
#  define _INDEX_HORIZONTAL_SLICE_
#  define _INDEX_HORIZONTAL_SLICE_PLUS_1_(i) (i)
#  define _INDEX_HORIZONTAL_SLICE_PLUS_2_(i,j) (i,j)
#  define _DIMENSION_HORIZONTAL_SLICE_AUTOMATIC_
#endif

! Preprocessor symbols for procedures operating on an INTERIOR slice
#ifdef _FABM_VECTORIZED_DIMENSION_INDEX_
!  Interior procedures operate in 1D
!  Interior slices MUST be 1D arrays; horizontal slices may be scalars or 1D arrays
#  define _LOOP_END_ end do
#  ifdef _HORIZONTAL_IS_VECTORIZED_
!    Horizontal slices are 1D arrays
!    For instance, model with i,j,k [MOM,NEMO] or i,j or i,k [FVCOM], or i; vectorized along i or j
#    define _DECLARE_INTERIOR_INDICES_ integer :: _I_,_J_
#    define _LOOP_BEGIN_EX_(cache) do _I_=1,cache%n;_J_=_I_
#    define _CONCURRENT_LOOP_BEGIN_EX_(cache) _DO_CONCURRENT_(_I_,1,cache%n);_J_=_I_
#  else
!    Horizontal slices are scalars
!    For instance model with k [GOTM] or i,k, i,j,k, vectorized along k
#    define _DECLARE_INTERIOR_INDICES_ integer :: _I_
#    define _LOOP_BEGIN_EX_(cache) do _I_=1,cache%n
#    define _CONCURRENT_LOOP_BEGIN_EX_(cache) _DO_CONCURRENT_(_I_,1,cache%n)
#  endif
#else
!  Interior procedures operate in 0D
!  Interior slices may be 0D scalars or 1D arrays [the latter if the model has a vertical dimension]
#  define _LOOP_BEGIN_EX_(cache)
#  define _CONCURRENT_LOOP_BEGIN_EX_(cache)
#  define _LOOP_END_
#  ifdef _INTERIOR_IS_VECTORIZED_
!    Interior slices are 1D arrays - we will operate on their first element (_I_=1)
#    define _DECLARE_INTERIOR_INDICES_ integer,parameter :: _I_=1
#  else
!    Interior slices are scalars
#    define _DECLARE_INTERIOR_INDICES_
#  endif
#endif
#define _ARGUMENTS_INTERIOR_ cache
#define _DECLARE_ARGUMENTS_INTERIOR_ type (type_interior_cache),intent(inout) :: cache;_DECLARE_INTERIOR_INDICES_
#define _LOOP_BEGIN_ _LOOP_BEGIN_EX_(cache)
#define _CONCURRENT_LOOP_BEGIN_ _CONCURRENT_LOOP_BEGIN_EX_(cache)

! Preprocessor symbols for procedures operating on a HORIZONTAL slice
#ifdef _HORIZONTAL_IS_VECTORIZED_
!  Horizontal procedures operate in 1D
!  Horizontal and interior slices MUST be 1D. Use same index for horizontal and interior (_I_=_J_)
#  define _DECLARE_HORIZONTAL_INDICES_ integer :: _I_,_J_
#  define _HORIZONTAL_LOOP_BEGIN_EX_(cache) do _J_=1,cache%n;_I_=_J_
#  define _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(cache) _DO_CONCURRENT_(_J_,1,cache%n);_I_=_J_
#  define _HORIZONTAL_LOOP_END_ end do
#else
!  Horizontal procedures operate in 0D
!  Horizontal slices MUST be scalars; interior slices can be scalars or 1D arrays
#  define _HORIZONTAL_LOOP_BEGIN_EX_(cache)
#  define _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(cache)
#  define _HORIZONTAL_LOOP_END_
#  ifdef _INTERIOR_IS_VECTORIZED_
!    Interior slices are 1D arrays - we will operate on their first element (_I_=1)
#    define _DECLARE_HORIZONTAL_INDICES_ integer,parameter :: _I_=1
#  else
!    Interior slices are scalars
#    define _DECLARE_HORIZONTAL_INDICES_
#  endif
#endif
#define _ARGUMENTS_HORIZONTAL_ cache
#define _DECLARE_ARGUMENTS_HORIZONTAL_ type (type_horizontal_cache),intent(inout) :: cache;_DECLARE_HORIZONTAL_INDICES_
#define _HORIZONTAL_LOOP_BEGIN_ _HORIZONTAL_LOOP_BEGIN_EX_(cache)
#define _CONCURRENT_HORIZONTAL_LOOP_BEGIN_ _CONCURRENT_HORIZONTAL_LOOP_BEGIN_EX_(cache)
#define _SURFACE_LOOP_BEGIN_ _HORIZONTAL_LOOP_BEGIN_
#define _SURFACE_LOOP_END_ _HORIZONTAL_LOOP_END_
#define _BOTTOM_LOOP_BEGIN_ _HORIZONTAL_LOOP_BEGIN_
#define _BOTTOM_LOOP_END_ _HORIZONTAL_LOOP_END_

! Preprocessor symbols for procedures operating on a VERTICAL slice
#ifdef _FABM_DEPTH_DIMENSION_INDEX_
!  Vertical procedures operate in 1D
!  Interior slices MUST be 1D arrays; horizontal slices may be 0D or 1D
!  Applies to all models with depth dimension. For instance, model with i,j,k [MOM,NEMO], i,k [FVCOM], or k [GOTM]
#  define _VERTICAL_LOOP_END_ end do
#  define _VERTICAL_LOOP_EXIT_ exit
#  ifdef _FABM_VERTICAL_BOTTOM_TO_SURFACE_
#    define _DOWNWARD_LOOP_BEGIN_ do _I_=_N_,1,-1
#    define _UPWARD_LOOP_BEGIN_ do _I_=1,_N_
#    define _MOVE_TO_BOTTOM_ _I_=1
#    define _MOVE_TO_SURFACE_ _I_=_N_
#  else
#    define _DOWNWARD_LOOP_BEGIN_ do _I_=1,_N_
#    define _UPWARD_LOOP_BEGIN_ do _I_=_N_,1,-1
#    define _MOVE_TO_SURFACE_ _I_=1
#    define _MOVE_TO_BOTTOM_ _I_=_N_
#  endif
#  define _CONCURRENT_VERTICAL_LOOP_BEGIN_EX_(cache) _DO_CONCURRENT_(_I_,1,cache%n)
#  ifdef _HORIZONTAL_IS_VECTORIZED_
!    Horizontal slices are 1D arrays - we will operate on their first element (_J_=1)
!    For instance, model with i,j,k [MOM,NEMO] or i,k [FVCOM]; vectorized along i or j
#    define _DECLARE_VERTICAL_INDICES_ integer :: _I_;integer,parameter :: _J_=1
#  else
!    Horizontal slices are scalars
!    For instance model with k [GOTM] or i,k, i,j,k, vectorized along k
#    define _DECLARE_VERTICAL_INDICES_ integer :: _I_
#  endif
#else
!  Vertical procedures operate in 0D
!  Interior slices may scalars or 1D arrays [the latter if the model is vectorized over a horizontal dimension]
!  Applies to all models without depth dimension; for instance, 0D box or model with i,j or i
#  define _CONCURRENT_VERTICAL_LOOP_BEGIN_EX_(cache)
#  define _VERTICAL_LOOP_END_
#  define _VERTICAL_LOOP_EXIT_
#  define _DOWNWARD_LOOP_BEGIN_
#  define _UPWARD_LOOP_BEGIN_
#  define _MOVE_TO_SURFACE_
#  define _MOVE_TO_BOTTOM_
#  ifdef _INTERIOR_IS_VECTORIZED_
!    Interior slices are 1D arrays. Since the vectorized dimension is not depth, horizontal slices MUST be 1D arrays too
!    Operate on their first element (_I_=1,_J_=1)
!    For instance, model with i,j or i; vectorized along i or j
#    define _DECLARE_VERTICAL_INDICES_ integer,parameter :: _I_=1,_J_=1
#  else
!    Both interior and horizontal slices are scalars
!    For instance, 0D box
#    define _DECLARE_VERTICAL_INDICES_
#  endif
#endif
#define _VERTICAL_LOOP_BEGIN_ _DOWNWARD_LOOP_BEGIN_
#define _DOWNWARD_LOOP_END_ _VERTICAL_LOOP_END_
#define _UPWARD_LOOP_END_ _VERTICAL_LOOP_END_
#define _ARGUMENTS_VERTICAL_ cache
#define _DECLARE_ARGUMENTS_VERTICAL_ type (type_vertical_cache),intent(inout) :: cache;_DECLARE_VERTICAL_INDICES_
#define _CONCURRENT_VERTICAL_LOOP_BEGIN_ _CONCURRENT_VERTICAL_LOOP_BEGIN_EX_(cache)

! Preprocessor symbols for procedures operating on a single point in space.
#ifdef _INTERIOR_IS_VECTORIZED_
#  ifdef _HORIZONTAL_IS_VECTORIZED_
#    define _ARGUMENTS_LOCAL_ cache,_I_,_J_
#    define _DECLARE_ARGUMENTS_LOCAL_ class (type_cache),intent(in) :: cache;integer,intent(in) :: _I_,_J_
#  else
#    define _ARGUMENTS_LOCAL_ cache,_I_
#    define _DECLARE_ARGUMENTS_LOCAL_ class (type_cache),intent(in) :: cache;integer,intent(in) :: _I_
#  endif
#else
#  define _ARGUMENTS_LOCAL_ cache
#  define _DECLARE_ARGUMENTS_LOCAL_ class (type_cache),intent(in) :: cache
#endif

! For BGC models: FABM arguments to routines implemented by biogeochemical models.
#define _ARGUMENTS_DO_ _ARGUMENTS_INTERIOR_
#define _ARGUMENTS_DO_PPDD_ _ARGUMENTS_INTERIOR_,pp,dd
#define _ARGUMENTS_DO_SURFACE_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_DO_BOTTOM_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_DO_BOTTOM_PPDD_ _ARGUMENTS_HORIZONTAL_,pp,dd,benthos_offset
#define _ARGUMENTS_DO_COLUMN_ _ARGUMENTS_VERTICAL_
#define _ARGUMENTS_GET_VERTICAL_MOVEMENT_ _ARGUMENTS_INTERIOR_
#define _ARGUMENTS_GET_EXTINCTION_ _ARGUMENTS_INTERIOR_
#define _ARGUMENTS_GET_DRAG_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_GET_ALBEDO_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_CHECK_STATE_ _ARGUMENTS_INTERIOR_
#define _ARGUMENTS_CHECK_SURFACE_STATE_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_CHECK_BOTTOM_STATE_ _ARGUMENTS_HORIZONTAL_
#define _ARGUMENTS_INITIALIZE_STATE_ _ARGUMENTS_INTERIOR_
#define _ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_ _ARGUMENTS_HORIZONTAL_

! For BGC models: Declaration of FABM arguments to routines implemented by biogeochemical models.
#define _DECLARE_ARGUMENTS_DO_  _DECLARE_ARGUMENTS_INTERIOR_
#define _DECLARE_ARGUMENTS_DO_PPDD_ _DECLARE_ARGUMENTS_INTERIOR_;real(rke) _DIMENSION_SLICE_PLUS_2_,intent(inout) :: pp,dd
#define _DECLARE_ARGUMENTS_DO_BOTTOM_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_DO_BOTTOM_PPDD_ _DECLARE_ARGUMENTS_HORIZONTAL_;real(rke) _DIMENSION_HORIZONTAL_SLICE_PLUS_2_,intent(inout) :: pp,dd;integer,intent(in) :: benthos_offset
#define _DECLARE_ARGUMENTS_DO_COLUMN_ _DECLARE_ARGUMENTS_VERTICAL_
#define _DECLARE_ARGUMENTS_DO_SURFACE_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_GET_VERTICAL_MOVEMENT_ _DECLARE_ARGUMENTS_INTERIOR_
#define _DECLARE_ARGUMENTS_GET_EXTINCTION_ _DECLARE_ARGUMENTS_INTERIOR_
#define _DECLARE_ARGUMENTS_GET_DRAG_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_GET_ALBEDO_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_CHECK_STATE_ _DECLARE_ARGUMENTS_INTERIOR_
#define _DECLARE_ARGUMENTS_CHECK_SURFACE_STATE_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_CHECK_BOTTOM_STATE_ _DECLARE_ARGUMENTS_HORIZONTAL_
#define _DECLARE_ARGUMENTS_INITIALIZE_STATE_ _DECLARE_ARGUMENTS_INTERIOR_
#define _DECLARE_ARGUMENTS_INITIALIZE_HORIZONTAL_STATE_ _DECLARE_ARGUMENTS_HORIZONTAL_

#define _ADD_(variable,value) cache%write _INDEX_SLICE_PLUS_1_(variable%sum_index) = cache%write _INDEX_SLICE_PLUS_1_(variable%sum_index) + (value)
#define _ADD_HORIZONTAL_(variable,value) cache%write_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_sum_index) = cache%write_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_sum_index) + (value)

! For BGC models: Expressions for setting space-dependent FABM variables defined on the full spatial domain.
#define _ADD_SOURCE_(variable,value) _ADD_(variable%sms,(value)*self%rdt__)
#define _ADD_BOTTOM_SOURCE_(variable,value) _ADD_HORIZONTAL_(variable%bottom_sms,(value)*self%rdt__)
#define _ADD_SURFACE_SOURCE_(variable,value) _ADD_HORIZONTAL_(variable%surface_sms,(value)*self%rdt__)
#define _ADD_BOTTOM_FLUX_(variable,value) _ADD_HORIZONTAL_(variable%bottom_flux,(value)*self%rdt__)
#define _ADD_SURFACE_FLUX_(variable,value) _ADD_HORIZONTAL_(variable%surface_flux,(value)*self%rdt__)
#define _SET_DD_(variable1,variable2,value) dd _INDEX_SLICE_PLUS_2_(variable1%state_index,variable2%state_index) = dd _INDEX_SLICE_PLUS_2_(variable1%state_index,variable2%state_index) + (value)*self%rdt__
#define _SET_PP_(variable1,variable2,value) pp _INDEX_SLICE_PLUS_2_(variable1%state_index,variable2%state_index) = pp _INDEX_SLICE_PLUS_2_(variable1%state_index,variable2%state_index) + (value)*self%rdt__
#define _ADD_VERTICAL_VELOCITY_(variable,value) _ADD_(variable%movement,(value)*self%rdt__)
#define _INVALIDATE_STATE_ cache%valid = .false.
#define _REPAIR_STATE_ cache%repair

#define _GET_WITH_BACKGROUND_(variable,target) target = cache%read _INDEX_SLICE_PLUS_1_(variable%index)+variable%background
#define _GET_HORIZONTAL_WITH_BACKGROUND_(variable,target) target = cache%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_index)+variable%background

! For BGC models: quick expressions for setting a single element in both the destruction and production matrix.
#define _SET_DD_SYM_(variable1,variable2,value) _SET_DD_(variable1,variable2,value);_SET_PP_(variable2,variable1,value)
#define _SET_PP_SYM_(variable1,variable2,value) _SET_PP_(variable1,variable2,value);_SET_DD_(variable2,variable1,value
)

! For BGC models: macro to determine whether a variable identifier is in use (i.e., has been registered with FABM)
#define _VARIABLE_REGISTERED_(variable) associated(variable%link)
#define _AVAILABLE_(variable) variable%index/=-1
#define _AVAILABLE_HORIZONTAL_(variable) variable%horizontal_index/=-1

! For BGC models: read/write variable access.
#define _GET_(variable,target) target = cache%read _INDEX_SLICE_PLUS_1_(variable%index)
#define _GET_HORIZONTAL_(variable,target) target = cache%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_index)
#define _GET_SURFACE_(variable,target) _GET_HORIZONTAL_(variable,target)
#define _GET_BOTTOM_(variable,target) _GET_HORIZONTAL_(variable,target)
#define _GET_GLOBAL_(variable,target) target = cache%read_scalar(variable%global_index)
#define _SET_(variable,value) cache%set_interior=.true.;cache%read _INDEX_SLICE_PLUS_1_(variable%index) = value
#define _SET_HORIZONTAL_(variable,value) cache%set_horizontal=.true.;cache%read_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_index) = value
#define _SET_DIAGNOSTIC_(variable,value) cache%write _INDEX_SLICE_PLUS_1_(variable%write_index) = value
#define _SET_HORIZONTAL_DIAGNOSTIC_(variable,value) cache%write_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%horizontal_write_index) = value
#define _SET_BOTTOM_DIAGNOSTIC_(variable,value) cache%write_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%bottom_write_index) = value
#define _SET_SURFACE_DIAGNOSTIC_(variable,value) cache%write_hz _INDEX_HORIZONTAL_SLICE_PLUS_1_(variable%surface_write_index) = value

#define _ASSERT_(condition, routine, message) if (.not.(condition)) call driver%fatal_error(routine, message)

! Backward compatibility with pre-1.0 FABM (2020-04-22)
#define _SET_ODE_(variable,value) _ADD_SOURCE_(variable,value)
#define _SET_BOTTOM_ODE_(variable,value) _ADD_BOTTOM_SOURCE_(variable,value)
#define _SET_SURFACE_ODE_(variable,value) _ADD_SURFACE_SOURCE_(variable,value)
#define _SET_BOTTOM_EXCHANGE_(variable,value) _ADD_BOTTOM_FLUX_(variable,value)
#define _SET_SURFACE_EXCHANGE_(variable,value) _ADD_SURFACE_FLUX_(variable,value)
#define _SET_VERTICAL_MOVEMENT_(variable,value) _ADD_VERTICAL_VELOCITY_(variable,value)
#define _SET_EXTINCTION_(value) _ADD_(self%extinction_id,value)
#define _SCALE_DRAG_(value) _ADD_HORIZONTAL_(self%surface_drag_id,(value)-1.0_rk)
#define _SET_ALBEDO_(value) _ADD_HORIZONTAL_(self%albedo_id,value)
```