```fortran
#include "fabm_driver.h"
#include "fabm_0d.h"
!-----------------------------------------------------------------------
!BOP
!
! !MODULE: 0D independent driver for the Framework for Aquatic Biogeochemical Models (FABM)
!
! !INTERFACE:
   module fabm_0d
!
! !DESCRIPTION:
! TODO
!
! !USES:
   use time
   use input
   use eqstate,only:rho_feistel

   use fabm
   use fabm_driver
   use fabm_expressions

   use shared
   use output

   implicit none
   private
!
! !PUBLIC MEMBER FUNCTIONS:
   public init_run, time_loop, clean_up
!
! !DEFINED PARAMETERS:
   integer, parameter :: namlst=10, yaml_unit=23
   integer, parameter :: CENTER=0,SURFACE=1,BOTTOM=2
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!  Run configuration file
   character(len=PATH_MAX) :: run_nml_file='run.nml'

!  FABM nml configuration file
   character(len=PATH_MAX) :: fabm_nml_file='fabm.nml'

!  FABM yaml configuration file
   character(len=PATH_MAX) :: fabm_yaml_file='fabm.yaml'

   ! Bio model info
   integer  :: ode_method
   logical  :: repair_state
   integer  :: swr_method
   logical  :: albedo_correction
   real(rk) :: cloud
   real(rk) :: par_fraction
   real(rk) :: par_background_extinction
   logical  :: apply_self_shading
   integer  :: model_type
   real(rk),allocatable :: current_rhs(:)

   real(rk), pointer :: bio_albedo, bio_extinction

   ! Shortcuts to number of state variables (interior, surface, bottom)
   integer :: n_int, n_sf, n_bt

   ! Environment
   real(rk),target :: current_depth,dens,decimal_yearday
   real(rk)        :: swr_sf,par_sf,par_bt,par_ct,extinction

   real(rk),allocatable :: expression_data(:)
   real(rk),allocatable :: totals0(:)

   type (type_scalar_input) :: mixing_rate
   type (type_scalar_input) :: mixed_layer_depth
   type (type_scalar_input), allocatable :: cc_deep(:)
   real(rk),allocatable,target :: w(:)
   integer(timestepkind), save :: itime

   type (type_fabm_interior_variable_id), save :: id_dens, id_par
   logical                                     :: compute_density

   type,extends(type_base_driver) :: type_fabm0d_driver
   contains
      procedure :: fatal_error => fabm0d_driver_fatal_error
      procedure :: log_message => fabm0d_driver_log_message
   end type
!EOP
!-----------------------------------------------------------------------

   contains

#define _ODE_ZEROD_
#include "ode_solvers_template.F90"

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Parse the command line
!
```

```
! !INTERFACE:
   subroutine cmdline

!   character(len=*), parameter :: version = '1.0'
   character(len=32) :: arg
   integer :: i
!EOP
!-----------------------------------------------------------------------
!BOC
   i=1
   do while (i <= command_argument_count())
      call get_command_argument(i, arg)

      select case (arg)
#if 0
      case ('-v', '--version')
         print '(2a)', 'fabm0d version ', RELEASE
         stop
#endif
      case ('-h', '--help')
         call print_help()
         stop
      case ('-r', '--run_nml')
         i = i+1
         call get_command_argument(i, run_nml_file)
#if 0
      case ('-n', '--nml')
         i = i+1
         call get_command_argument(i, fabm_nml_file)
#endif
      case ('-y', '--yaml')
         i = i+1
         call get_command_argument(i, fabm_yaml_file)
      case default
         print '(a,a,/)', 'Unrecognized command-line option: ', arg
         call print_help()
         stop
      end select
      i = i+1
   end do
#if 0
   print '(a)', trim(run_nml_file)
   print '(a)', trim(fabm_nml_file)
   print '(a)', trim(fabm_yaml_file)
#endif

   contains

   subroutine print_help()
      print '(a)', 'usage: fabm0d [OPTIONS]'
      print '(a)', ''
      print '(a)', 'Without further options, fabm0d run using default input filenames.'
      print '(a)', ''
      print '(a)', 'fabm0d options:'
      print '(a)', ''
      print '(a)', '  -h, --help        print usage information and exit'
      print '(a)', '  -r, --run_nml     namelist file with simualtion settings - default run.nml'
      print '(a)', '  -y, --yaml file   yaml-formatted file FABM configuration - default fabm.yaml'
      print '(a)', ''
   end subroutine print_help

   end subroutine  cmdline

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Initialise the model
!
! !INTERFACE:
   subroutine init_run()
!
! !DESCRIPTION:
!  This internal routine triggers the initialization of the model.
!  The first section reads the namelists of {\tt run.nml} with
!  the user specifications. Then, one by one each of the modules are
!  initialised.
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!
! !LOCAL VARIABLES:
   character(len=PATH_MAX)   :: env_file
   real(rk)                  :: depth, dt
   real(rk),parameter        :: invalid_latitude = -100._rk,invalid_longitude = -400.0_rk
   logical                   :: file_exists
   integer                   :: ios

   namelist /model_setup/ title,start,stop,dt,ode_method,repair_state,model_type
   namelist /environment/ env_file,swr_method,albedo_correction, &
                          latitude,longitude,cloud,par_fraction, &
                          depth,par_background_extinction,apply_self_shading
!EOP
!-----------------------------------------------------------------------
!BOC

   ! Make FABM use our custom logger/error reporter
   allocate(type_fabm0d_driver::driver)

   call cmdline

   LEVEL1 'init_run'
   STDERR LINE

   ! Open the namelist file.
   LEVEL2 'reading model setup namelists from ',trim(run_nml_file)
   open(namlst,file=run_nml_file,status='old',action='read',iostat=ios)
   if (ios/=0) call fatal_error('init_run','I could not open '//trim(run_nml_file)//' for reading.')

   ! Initialize environment
   temp%value = 0.0_rk
   salt%value = 0.0_rk
   light%value = 0.0_rk
   dens = 0.0_rk
   par_sf = 0.0_rk
   par_bt = 0.0_rk
   par_ct = 0.0_rk
   decimal_yearday = 0.0_rk
   model_type = 0

   ! Read all namelists
   title = ''
   start = ''
   stop = ''
   dt = 0.0_rk
   ode_method = 1
   repair_state = .false.
   read(namlst,nml=model_setup,iostat=ios)
   if (ios/=0) call fatal_error('init_run','I could not read the "model_setup" namelist from '//trim(run_nml_file)//'
.')

   ! Read environment namelist
   env_file = ''
   swr_method = 0
   albedo_correction = .true.
   latitude = invalid_latitude
   longitude = invalid_longitude
   cloud = 0.0_rk
   par_fraction = 1.0_rk
   depth = -1.0_rk
   par_background_extinction = 0.0_rk
   apply_self_shading = .true.
   read(namlst,nml=environment,iostat=ios)
   if (ios/=0) call fatal_error('init_run','I could not read the "environment" namelist from '//trim(run_nml_file)//'
.')

   compute_conserved_quantities = .false.
   call configure_output(namlst)

   ! Close the namelist file.
   close (namlst)

   if (start=='')  call fatal_error('init_run',trim(run_nml_file)//': start time "start" must be set in "model_setup"
 namelist.')
   if (stop=='')   call fatal_error('init_run',trim(run_nml_file)//': stop time "stop" must be set in "model_setup" n
amelist.')
   if (dt<=0.0_rk) call fatal_error('init_run',trim(run_nml_file)//': time step "dt" must be set to a positive value
in "model_setup" namelist.')
   if (env_file=='') call fatal_error('init_run',trim(run_nml_file)//': "env_file" must be set to a valid file path i
n "environment" namelist.')
   if (latitude/=invalid_latitude.and.(latitude<-90._rk.or.latitude>90._rk)) &
      call fatal_error('init_run',trim(run_nml_file)//': latitude must lie between -90 and 90.')
   if (longitude/=invalid_longitude.and.(longitude<-360._rk.or.longitude>360._rk)) &
      call fatal_error('init_run',trim(run_nml_file)//': longitude must lie between -360 and 360.')

   ! Make sure depth has been provided.
   if (depth<=0.0_rk) call fatal_error('init_run',trim(run_nml_file)//': &
      &a positive value for "depth" must be provided in "environment" namelist.')
   column_depth = depth ! Provided depth is the column depth. The modelled biogeochemistry will be positioned at half
 this depth.
   call update_depth(CENTER)

   ! If longitude and latitude are used, make sure they have been provided and are valid.
   if (swr_method==0) then
      if (latitude==invalid_latitude) call fatal_error('init_run',trim(run_nml_file)//': &
         &a valid value for "latitude" must be provided in "environment" if "swr_method" is 0.')
      if (longitude==invalid_longitude) call fatal_error('init_run',trim(run_nml_file)//': &
         &a valid value for "longitude" must be provided in "environment" if "swr_method" is 0.')
   end if

   ! Configure the time module to use actual start and stop dates.
   timefmt = 2

   ! Transfer the time step to the time module.
   timestep = dt

   ! Write information for this run to the console.
   LEVEL2 'Simulation: '//trim(title)
   select case (swr_method)
      case (0)
         LEVEL2 'Surface photosynthetically active radiation will be calculated from time,'
         LEVEL2 'cloud cover, and the simulated location at (lat,long)'
         LEVEL2 latitude,longitude
         LEVEL2 'Local PAR will be calculated from the surface value,'
         LEVEL2 'depth, and light extinction coefficient.'
         LEVEL2 'albedo_correction =',albedo_correction
```

```fortran
      case (1)
         LEVEL2 'Surface photosynthetically active radiation (PAR) is provided as input.'
         LEVEL2 'Local PAR will be calculated from the surface value,'
         LEVEL2 'depth, and light extinction coefficient.'
      case (2)
         LEVEL2 'Local photosynthetically active radiation is provided as input.'
   end select

   LEVEL2 'initializing modules....'

   ! Initialize the time module.
   call init_time(MinN,MaxN)

   ! Open the file with observations of the local environment.
   LEVEL1 'init environment'
   LEVEL2 'reading local environment data from:'
   LEVEL3 trim(env_file)
   call init_input()
   call light%configure(method=2, path=env_file, index=1, name='shortwave radiation')
   call temp%configure(method=2, path=env_file, index=2, name='temperature')
   call salt%configure(method=2, path=env_file, index=3, name='salinity')
   call register_input(light)
   call register_input(temp)
   call register_input(salt)

   ! Build FABM model tree. Use 'fabm_yaml_file' if available, otherwise fall back to fabm.nml.
   LEVEL1 'initialize FABM'
   LEVEL2 'reading configuration from:'
   inquire(file=trim(fabm_yaml_file),exist=file_exists)
   if (.not. file_exists) call fatal_error('init_run','can not find '//trim(fabm_yaml_file)//'.')
   LEVEL3 trim(fabm_yaml_file)
   model => fabm_create_model(path=trim(fabm_yaml_file))

   ! Shortcuts to the number of state variables.
   n_int = size(model%interior_state_variables)
   n_sf  = size(model%surface_state_variables)
   n_bt  = size(model%bottom_state_variables)

   allocate(cc(n_int+n_bt+n_sf))

   if (model_type==1) then
      call driver%log_message('The model type is set to mixed layer model (model_type = 1).')
      call driver%log_message('Therefore, bottom-associated processes will be deactivated.')
      allocate(cc_deep(n_int))
      cc_deep(:)%value = 0.0_rk
      mixing_rate%value = 0.0_rk
      allocate(w(n_int))
   end if

   ! Allocate memory to hold totals of conserved quantities
   allocate(totals0             (size(model%conserved_quantities)))  ! at initial time (depth-integrated, interior +
interfaces)
   allocate(totals              (size(model%conserved_quantities)))  ! at current time (depth-explicit, interior only
)
   allocate(int_change_in_totals(size(model%conserved_quantities)))  ! change since start of simulation (depth-integr
ated, interior + interfaces)

   call register_output_fields()

   ! Send information on spatial domain to FABM (this also allocates memory for diagnostics)
   call model%set_domain(seconds_per_time_unit=timestep)

   ! Create state variable vector, using the initial values specified by the model,
   ! and link state data to FABM.
   call model%link_all_interior_state_data(cc(1:n_int))
   call model%link_all_bottom_state_data  (cc(n_int+1:n_int+n_bt))
   call model%link_all_surface_state_data (cc(n_int+n_bt+1:n_int+n_bt+n_sf))

   id_dens = model%get_interior_variable_id(fabm_standard_variables%density)
   compute_density = model%variable_needs_values(id_dens)
   if (compute_density) call model%link_interior_data(id_dens,dens)

   id_par = model%get_interior_variable_id(fabm_standard_variables%downwelling_photosynthetic_radiative_flux)

   ! Link environmental data to FABM
   call model%link_interior_data(fabm_standard_variables%temperature,temp%value)
   call model%link_interior_data(fabm_standard_variables%practical_salinity,salt%value)
   if (model%variable_needs_values(id_par)) call model%link_interior_data(id_par,par)
   call model%link_interior_data(fabm_standard_variables%pressure,current_depth)
   call model%link_interior_data(fabm_standard_variables%cell_thickness,column_depth)
   call model%link_interior_data(fabm_standard_variables%depth,current_depth)
   call model%link_interior_data(fabm_standard_variables%attenuation_coefficient_of_photosynthetic_radiative_flux,ext
inction)
   call model%link_horizontal_data(fabm_standard_variables%surface_downwelling_photosynthetic_radiative_flux,par_sf)
   call model%link_horizontal_data(fabm_standard_variables%surface_downwelling_shortwave_flux,swr_sf)
   call model%link_horizontal_data(fabm_standard_variables%cloud_area_fraction,cloud)
   call model%link_horizontal_data(fabm_standard_variables%bottom_depth,column_depth)
   call model%link_horizontal_data(fabm_standard_variables%bottom_depth_below_geoid,column_depth)
   if (latitude /=invalid_latitude ) call model%link_horizontal_data(fabm_standard_variables%latitude,latitude)
   if (longitude/=invalid_longitude) call model%link_horizontal_data(fabm_standard_variables%longitude,longitude)
   call model%link_scalar(fabm_standard_variables%number_of_days_since_start_of_the_year,decimal_yearday)

   ! Read forcing data specified in input.yaml.
   call init_input_from_file('input.yaml')

   ! Request computation of contributions by BGC models to surface albedo and light attenuation
   call model%require_interior_data(fabm_standard_variables%attenuation_coefficient_of_photosynthetic_radiative_flux)
   call model%require_horizontal_data(fabm_standard_variables%surface_albedo)

   ! Check whether all dependencies of biogeochemical models have now been fulfilled.
   call model%start()

   ! Get pointers to contributions by BGC models to surface albedo and light attenuation
   bio_extinction => model%get_interior_data(model%get_interior_variable_id(fabm_standard_variables%attenuation_coeff
icient_of_photosynthetic_radiative_flux))
   bio_albedo => model%get_horizontal_data(model%get_horizontal_variable_id(fabm_standard_variables%surface_albedo))

   ! Update time and all time-dependent inputs.
   call update_environment(0_timestepkind)

   ! Perform custom initialization per biogeochemical model
   call model%initialize_interior_state()
   call model%initialize_surface_state()
   call model%initialize_bottom_state()

   ! Let FABM update the light field (requires state variables to be initialized!)
   call update_light()

   ! Allow the model to compute all diagnostics, so output for initial time contains sensible values.
   allocate(current_rhs(size(cc)))
   call get_rhs(.false.,size(cc),cc,current_rhs)
   call model%link_all_interior_state_data(cc(1:n_int))
   call model%link_all_bottom_state_data  (cc(n_int+1:n_int+n_bt))
   call model%link_all_surface_state_data (cc(n_int+n_bt+1:n_int+n_bt+n_sf))

   call get_conserved_quantities(totals0)
   int_change_in_totals = 0.0_rk

   LEVEL1 'init_output'
   call init_output(start)

   call do_output(0_timestepkind)

   STDERR LINE

   end subroutine init_run
!EOC

   subroutine init_input_from_file(path)
      use yaml_types
      use yaml,yaml_parse=>parse,yaml_error_length=>error_length

      character(len=*),intent(in) :: path

      logical                            :: exists
      character(len=yaml_error_length)   :: yaml_error
      class (type_node),         pointer :: root

      ! Determine whether input configuration file exists. If not, return.
      inquire(file=path,exist=exists)
      if (.not.exists) return

      ! Parse YAML.
      root => yaml_parse(path,yaml_unit,yaml_error)
      if (yaml_error/='') call driver%fatal_error('init_input_from_file',trim(yaml_error))

      ! Process root-level dictionary.
      select type (root)
         class is (type_dictionary)
         call init_input_from_yaml_node(root)
         class default
         call fatal_error('init_input_from_file',trim(path)//' must contain a dictionary with (variable name : inform
ation) pairs,&
               & not a single value.')
      end select
   end subroutine init_input_from_file

   subroutine init_input_from_yaml_node(mapping)
      use yaml_types

      class (type_dictionary),intent(in)  :: mapping

      character(len=64)                  :: variable_name
      type (type_key_value_pair),pointer :: pair
      integer                            :: i
      logical                            :: found

      pair => mapping%first
      if (associated(pair)) call driver%log_message('Forcing data specified in input.yaml:')
      do while (associated(pair))
         variable_name = trim(pair%key)
         if (variable_name=='') call driver%fatal_error('init_input_from_yaml_node','Empty variable name specified.')

         found = .false.
         if (model_type==1) then
            select case (variable_name)
            case ('mixed_layer_depth')
               call parse_input_variable(pair%key,pair%value,mixed_layer_depth)
               found = .true.
            case ('mixing_rate')
               call parse_input_variable(pair%key,pair%value,mixing_rate)
               found = .true.
            case default
               do i=1,n_int
                  if (variable_name=='deep/'//trim(model%interior_state_variables(i)%path)) then
```

```fortran
                     call parse_input_variable(pair%key,pair%value,cc_deep(i))
                     found = .true.
                  end if
               end do
            end select
         end if
         if (.not.found) call parse_input_variable(pair%key,pair%value)
         pair => pair%next
      end do
   end subroutine init_input_from_yaml_node

   subroutine parse_input_variable(variable_name,value_node,input_)
      use yaml_types

      character(len=*),        intent(in) :: variable_name
      class (type_node),target,intent(in) :: value_node
      type (type_scalar_input), target, optional :: input_

      class (type_dictionary),   pointer :: mapping
      type (type_error),         pointer :: config_error
      class (type_node),         pointer :: node
      class (type_scalar),       pointer :: constant_value_node, file_node
      real(rk)                           :: relaxation_time
      logical                            :: is_state_variable
      type (type_key_value_pair),pointer :: pair
      type (type_fabm_interior_variable_id)   :: interior_id
      type (type_fabm_horizontal_variable_id) :: horizontal_id
      type (type_fabm_scalar_variable_id)     :: scalar_id
      type (type_scalar_input), pointer  :: input

      select type (value_node)
      class is (type_dictionary)
         mapping => value_node
      class default
         call fatal_error('init_input_from_yaml_node','Contents of '//trim(value_node%path)//' must be a dictionary,
not a single value.')
      end select

      is_state_variable = .false.
      if (present(input_)) then
         input => input_
      else
         allocate(input)
         call extra_inputs%add(input)

         ! Try to locate the forced variable among interior, horizontal, and global variables in the active biogeoche
mical models.
         interior_id = model%get_interior_variable_id(variable_name)
         if (model%is_variable_used(interior_id)) then
            is_state_variable = interior_id%variable%state_indices%value/=-1
         else
            horizontal_id = model%get_horizontal_variable_id(variable_name)
            if (model%is_variable_used(horizontal_id)) then
               is_state_variable = horizontal_id%variable%state_indices%value/=-1
            else
               scalar_id = model%get_scalar_variable_id(variable_name)
               if (.not. model%is_variable_used(scalar_id)) call log_message('WARNING! input.yaml: &
                  &Variable "'//trim(variable_name)//'" is not present in any biogeochemical model.')
            end if
         end if
      end if

      ! Prepend to list of input data.
      input%name = trim(variable_name)

      constant_value_node => mapping%get_scalar('constant_value',required=.false.,error=config_error)
      file_node => mapping%get_scalar('file',required=.false.,error=config_error)
      if (associated(constant_value_node)) then
         ! Input variable is set to a constant value.
         input%method = 0
         input%constant_value = mapping%get_real('constant_value',error=config_error)
         if (associated(config_error)) call fatal_error('parse_input_variable',config_error%message)

         ! Make sure keys related to time-varying input are not present.
         if (associated(file_node)) call fatal_error('parse_input_variable','input.yaml, variable "'//trim(variable_n
ame)//'": &
            &keys "constant_value" and "file" cannot both be present.')
         node => mapping%get('column')
         if (associated(node)) call fatal_error('parse_input_variable','input.yaml, variable "'//trim(variable_name)/
/'": &
            &keys "constant_value" and "column" cannot both be present.')
         node => mapping%get('scale_factor')
         if (associated(node)) call fatal_error('parse_input_variable','input.yaml, variable "'//trim(variable_name)/
/'": &
            &keys "constant_value" and "scale_factor" cannot both be present.')
      elseif (associated(file_node)) then
         ! Input variable is set to a time-varying value. Obtain path, column number and scale factor.
         input%method = 2
         input%path = mapping%get_string('file',error=config_error)
         if (associated(config_error)) call fatal_error('parse_input_variable',config_error%message)
         input%index = mapping%get_integer('column',default=1,error=config_error)
         if (associated(config_error)) call fatal_error('parse_input_variable',config_error%message)
         input%scale_factor = mapping%get_real('scale_factor',default=1.0_rk,error=config_error)
         if (associated(config_error)) call fatal_error('parse_input_variable',config_error%message)
      else
         call fatal_error('parse_input_variable','input.yaml, variable "'//trim(variable_name)//'": &
            &either key "constant_value" or key "file" must be present.')
      end if
      call register_input(input)

      if (is_state_variable) then
         ! This is a state variable. Obtain associated relaxation time.
         relaxation_time = mapping%get_real('relaxation_time',default=1.e15_rk,error=config_error)
         if (associated(config_error)) call fatal_error('parse_input_variable',config_error%message)
      else
         ! This is not a state variable. Make sure no relaxation time is specified.
         node => mapping%get('relaxation_time')
         if (associated(node)) call fatal_error('parse_input_variable','input.yaml, variable "'//trim(variable_name)/
/'": &
            &key "relaxation_time" is not supported because "'//trim(variable_name)//'" is not a state variable.')
      end if

      ! Warn about uninterpreted keys.
      pair => mapping%first
      do while (associated(pair))
         if (.not.pair%accessed) call fatal_error('parse_input_variable','input.yaml: &
            &Unrecognized option "'//trim(pair%key)//'" found for variable "'//trim(variable_name)//'".')
         pair => pair%next
      end do

      ! If a data pointer was provided, this variable for the host, not FABM, so return.
      if (present(input_)) return

      ! Link forced data to target variable.
      if (model%is_variable_used(interior_id)) then
         call model%link_interior_data(interior_id, input%value, source=data_source_user)
      elseif (model%is_variable_used(horizontal_id)) then
         call model%link_horizontal_data(horizontal_id, input%value, source=data_source_user)
      else
         call model%link_scalar(scalar_id, input%value, source=data_source_user)
      end if

   end subroutine parse_input_variable

   subroutine update_environment(n)
      integer(timestepkind),intent(in) :: n

      ! Update time in time manager
      call update_time(n)

      ! Compute decimal year day (input for some biogeochemical models)
      decimal_yearday = yearday-1 + dble(secondsofday)/86400

      ! Update environment (i.e., read from input files)
      call do_input(julianday,secondsofday)

      if (model_type==1) column_depth = mixed_layer_depth%value

      ! Compute density from temperature and salinity, if required by biogeochemistry.
      if (compute_density) dens = rho_feistel(salt%value,temp%value,0.5_rk*column_depth,.true.)
   end subroutine update_environment

   subroutine update_light()
      real(rk) :: zenith_angle,solar_zenith_angle
      real(rk) :: shortwave_radiation
      real(rk) :: albedo,albedo_water
      real(rk) :: hh

      ! Calculate photosynthetically active radiation at surface, if it is not provided in the input file.
      if (swr_method==0) then
         ! Calculate photosynthetically active radiation from geographic location, time, cloud cover.
         hh = secondsofday*(1._rk/3600)
         zenith_angle = solar_zenith_angle(yearday,hh,longitude,latitude)
         swr_sf = shortwave_radiation(zenith_angle,yearday,longitude,latitude,cloud)
         if (albedo_correction) then
            albedo = albedo_water(1,zenith_angle,yearday)
            swr_sf = swr_sf*(1._rk-albedo-bio_albedo)
         end if
      else
         swr_sf = light%value
      end if

      ! Multiply by fraction of short-wave radiation that is photosynthetically active.
      par_sf = par_fraction*swr_sf

      ! Apply light attentuation with depth, unless local light is provided in the input file.
      if (swr_method/=2) then
         ! Calculate light extinction
         extinction = 0.0_rk
         if (apply_self_shading) extinction = bio_extinction
         extinction = extinction + par_background_extinction

         ! Either we calculate surface PAR, or surface PAR is provided.
         ! Calculate the local PAR at the given depth from par fraction, extinction coefficient, and depth.
         par_ct = par_sf*exp(-0.5_rk*column_depth*extinction)
         par_bt = par_sf*exp(-column_depth*extinction)
      else
         par_ct = par_sf
         par_bt = par_sf
      end if
      call update_depth(CENTER)

   end subroutine update_light

!-----------------------------------------------------------------------
!BOP
```

```
!
! !IROUTINE: Manage global time--stepping \label{timeLoop}
!
! !INTERFACE:
   subroutine time_loop()
!
! !DESCRIPTION:
! This internal routine is the heart of the code. It contains
! the main time-loop inside of which all routines required
! during the time step are called.
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
! !LOCAL VARIABLES:
   logical                   :: valid_state
   integer                   :: progress,k
!EOP
!-----------------------------------------------------------------------
!BOC
   LEVEL1 'time_loop'

   progress = (MaxN-MinN+1)/10
   k = 0
   do itime=MinN,MaxN
      if(mod(itime,progress) == 0 .or. itime == MinN) then
         LEVEL0 k,'%'
         k = k+10
      end if

      ! Update time and all time-dependent inputs.
      call update_environment(itime)
      call update_light()

      ! Integrate one time step
      call ode_solver(ode_method,size(cc),timestep,cc,get_rhs,get_ppdd)
      call get_rhs(.false.,size(cc),cc,current_rhs)

      ! ODE solver may have redirected the current state to an array with intermediate values.
      ! Reset to global array.
      call model%link_all_interior_state_data(cc(1:n_int))
      call model%link_all_bottom_state_data  (cc(n_int+1:n_int+n_bt))
      call model%link_all_surface_state_data (cc(n_int+n_bt+1:n_int+n_sf+n_bt))

      ! Verify whether the model state is still valid (clip if needed and allowed)
      call model%check_interior_state(repair_state,valid_state)
      if (valid_state .or. repair_state) call model%check_bottom_state(repair_state,valid_state)
      if (valid_state .or. repair_state) call model%check_surface_state(repair_state,valid_state)
      if (.not. (valid_state .or. repair_state)) &
         call fatal_error('time_loop','State variable values are invalid and repair is not allowed. &
            &This may be fixed by setting repair_state=.true. (clip state to nearest valid value), &
            &but this should be used with caution. Try and decrease the time step (dt) first - and see if that helps.
')

      if (compute_conserved_quantities) then
         call get_conserved_quantities(int_change_in_totals)
         int_change_in_totals = int_change_in_totals - totals0
      end if

      ! Do output
      call do_output(itime)
   end do
   STDERR LINE

   end subroutine time_loop
!EOC

   subroutine get_conserved_quantities(depth_int_totals)
      real(rk), intent(inout) :: depth_int_totals(size(model%conserved_quantities))
      real(rk) :: totals_hz(size(model%conserved_quantities))
      call model%get_interior_conserved_quantities(totals)
      call model%get_horizontal_conserved_quantities(totals_hz)
      depth_int_totals = totals*column_depth + totals_hz
   end subroutine
!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: The run is over --- now clean up.
!
! !INTERFACE:
   subroutine clean_up(ignore_errors)
!
! !DESCRIPTION:
! Close all open files.
!
! !INPUT PARAMETERS:
   logical, intent(in) :: ignore_errors
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!-----------------------------------------------------------------------
!BOC
   LEVEL1 'clean_up'

   call close_input()
   call clean_output(ignore_errors=ignore_errors)
```

```fortran

   end subroutine clean_up
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Get the right-hand side of the OĐE system.
!
! !INTERFACE:
   subroutine update_depth(location)
!
! !ĐESCRIPTION:
! TOĐO
!
! !INPUT PARAMETERS:
   integer, intent(in)                  :: location
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!-----------------------------------------------------------------------
!BOC
   select case (location)
      case (SURFACE)
         current_depth = 0.0_rk
         par = par_sf
      case (BOTTOM)
         current_depth = column_depth
         par = par_bt
      case (CENTER)
         current_depth = 0.5_rk*column_depth
         par = par_ct
   end select

   end subroutine update_depth
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Get the right-hand side of the OĐE system.
!
! !INTERFACE:
   subroutine get_ppdd(first,numc,cc,pp,dd)
!
! !ĐESCRIPTION:
! TOĐO
!
! !INPUT PARAMETERS:
   logical, intent(in)                  :: first
   integer, intent(in)                  :: numc
   real(rk), intent(in)                 :: cc(1:numc)
!
! !OUTPUT PARAMETERS:
   real(rk), intent(out)                :: pp(1:numc,1:numc)
   real(rk), intent(out)                :: dd(1:numc,1:numc)
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
! !LOCAL PARAMETERS:
!EOP
!-----------------------------------------------------------------------
!BOC
   ! Initialize production/destruction matrices to zero (entries will be incremented by FABM)
   pp = 0.0_rk
   dd = 0.0_rk

   ! Send current state to FABM
   ! (this may differ from the global state cc if using a multi-step integration scheme such as Runge-Kutta)
   call model%link_all_interior_state_data(cc(1:n_int))
   call model%link_all_bottom_state_data  (cc(n_int+1:n_int+n_bt))
   call model%link_all_surface_state_data (cc(n_int+n_bt+1:n_int+n_bt+n_sf))

   call model%prepare_inputs(real(itime,rk))

   ! Calculate temporal derivatives due to benthic processes.
   call update_depth(BOTTOM)
   call model%get_bottom_sources(pp,dd,n_int)

   ! For pelagic variables: translate bottom flux to into change in concentration
   pp(1:n_int,:) = pp(1:n_int,:)/column_depth
   dd(1:n_int,:) = dd(1:n_int,:)/column_depth

   ! For pelagic variables: surface and bottom flux (rate per surface area) to concentration (rate per volume)
   call update_depth(CENTER)
   call model%get_interior_sources(pp,dd)

   call model%finalize_outputs()

   end subroutine get_ppdd
!EOC

!-----------------------------------------------------------------------
!BOP
!
```

```
! !IROUTINE: Get the right-hand side of the ODE system.
!
! !INTERFACE:
   subroutine get_rhs(first,numc,cc,rhs)
!
! !DESCRIPTION:
! TODO
!
! !INPUT PARAMETERS:
   logical, intent(in)                  :: first
   integer, intent(in)                  :: numc
   real(rk), intent(in)                 :: cc(1:numc)
!
! !OUTPUT PARAMETERS:
   real(rk), intent(out)                :: rhs(1:numc)
!
! !LOCAL PARAMETERS:
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!-----------------------------------------------------------------------
!BOC
   if (first) then
      rhs = current_rhs
      return
   end if

   ! Initialize derivatives to zero (entries will be incremented by FABM)
   rhs = 0.0_rk

   ! Send current state to FABM
   ! (this may differ from the global state cc if using a multi-step integration scheme such as Runge-Kutta)
   call model%link_all_interior_state_data(cc(1:n_int))
   call model%link_all_bottom_state_data  (cc(n_int+1:n_int+n_bt))
   call model%link_all_surface_state_data (cc(n_int+n_bt+1:n_int+n_bt+n_sf))

   call model%prepare_inputs(real(itime,rk))

   ! Calculate temporal derivatives due to surface-bound processes.
   call update_depth(SURFACE)
   call model%get_surface_sources(rhs(1:n_int),rhs(n_int+n_bt+1:n_int+n_bt+n_sf))

   ! Calculate temporal derivatives due to bottom-bound processes.
   select case (model_type)
   case (0)
      call update_depth(BOTTOM)
      call model%get_bottom_sources(rhs(1:n_int),rhs(n_int+1:n_int+n_bt))
   case (1)
      call model%get_vertical_movement(w)
      rhs(1:n_int) = rhs(1:n_int) + mixing_rate%value * (cc_deep(1:n_int)%value - cc(1:n_int)) + w * cc(1:n_int)
   end select

   ! For pelagic variables: surface and bottom flux (rate per surface area) to concentration (rate per volume)
   rhs(1:n_int) = rhs(1:n_int)/column_depth

   ! Add change in pelagic variables.
   call update_depth(CENTER)
   call model%get_interior_sources(rhs(1:n_int))

   call model%finalize_outputs()

   end subroutine get_rhs
!EOC

   subroutine fabm0d_driver_fatal_error(self,location,message)
      class (type_fabm0d_driver), intent(inout) :: self
      character(len=*),       intent(in)    :: location,message

      write (stderr,'(A)') ''
      write (stderr,'(A)') 'FATAL ERROR: '//trim(location)
      write (stderr,'(A)') trim(message)
      call clean_up(ignore_errors=.true.)
      stop 1
   end subroutine

   subroutine fabm0d_driver_log_message(self,message)
      class (type_fabm0d_driver), intent(inout) :: self
      character(len=*),       intent(in)    :: message

      write (stdout,'(A)') trim(message)
   end subroutine

!-----------------------------------------------------------------------

   end module fabm_0d

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