```
#include "fabm_driver.h"
#include "fabm.h"

#define STDERR write(*,*)
#define LEVEL0 STDERR
#define LEVEL1 STDERR '   ',
#define LEVEL2 STDERR '       ',
#define LEVEL3 STDERR '           ',
#define LEVEL4 STDERR '               ',
#define FATAL  STDERR 'FATAL ERROR: ',

!-----------------------------------------------------------------------
!BOP
!
! !MODULE: fabm_iow_spm --- 1-class SPM model,
!
! !INTERFACE:
   module fabm_iow_spm
!
! !USES:
   use fabm_types

!  default: all is private.
   private
!
! !PRIVATE DATA MEMBERS:
!
! !REVISION HISTORY:!
!  Original author(s): Manuel Ruiz Villarreal & Richard Hofmeister & Ulf Gräwe
!
!
! !PUBLIC DERIVED TYPES:
   type, extends(type_base_model),public :: type_iow_spm
!     Variable identifiers
      type (type_state_variable_id)                  :: id_spm      ! concentrations
      type (type_bottom_state_variable_id)           :: id_pmpool   !sediment pool
      type (type_horizontal_dependency_id)           :: id_taub     !bottom stress
      type (type_dependency_id)                      :: id_temp     !temperature
      type (type_dependency_id)                      :: id_rhow     !density
      type (type_horizontal_diagnostic_variable_id)  :: id_massflux !exchange layer
      type (type_horizontal_diagnostic_variable_id)  :: id_bedload  !bedload
      type (type_horizontal_diagnostic_variable_id)  :: id_massfraction  !
      type (type_horizontal_dependency_id)           :: id_total_pmpool
      ! van Kesse L2 model
      type (type_bottom_state_variable_id)           :: id_fluffayer  !sediment pool
      type (type_horizontal_dependency_id)           :: id_total_fluffpool
      type (type_horizontal_dependency_id)           :: id_total_mudpool
      ! provide information of the light model
      type (type_diagnostic_variable_id)             :: id_dPAR
      type (type_dependency_id)                      :: id_par

!     Model parameters
      real(rk) :: diameter
      real(rk) :: c_init
      real(rk) :: thickness_L1
      real(rk) :: thickness_fluff
      real(rk) :: tauc_factor
      real(rk) :: M0
      real(rk) :: M1
      real(rk) :: M2
      real(rk) :: flux_alpha
      logical  :: cohesive
      integer  :: sinking_method
```

```
      integer  :: bottom_stress_method
      logical  :: pm_pool
      real(rk) :: shading
      real(rk) :: tauc_const
      real(rk) :: ws_const
      real(rk) :: rho
      logical  :: consolidate_bed
      integer  :: consolidate_bed_method
      integer  :: bedload_method
      real(rk) :: bedload_factor
      logical  :: add_to_density
      integer  :: resuspension_model
      real(rk) :: morfac
      logical  :: sand_mud_interaction
      real(rk) :: crit_mud
      real(rk) :: stressexponent
      logical  :: use_par

      contains

      procedure :: initialize
      procedure :: do
      procedure :: do_bottom
      procedure :: get_vertical_movement
      procedure :: get_light_extinction

   end type type_iow_spm
!EOP
!-----------------------------------------------------------------------

   contains

!-----------------------------------------------------------------------
!BOP
!
! !ROUTINE: Initialise the sediment model
!
! !INTERFACE:
   subroutine initialize(self,configunit)
!
! !DESCRIPTION:
!  Here, the spm namelist is read and te variables exported
!  by the model are registered with FABM.
!
! !USES:
   implicit none
!
! !INPUT PARAMETERS:

   integer,                         intent(in)            :: configunit
   class (type_iow_spm),            intent(inout), target :: self

!
! !REVISION HISTORY:
!  Original author(s): Ulf Gräwe & Richard Hofmeister
!
! !LOCAL VARIABLES:

   real(rk) :: diameter=100.0_rk         ! mikrons
   real(rk) :: c_init=5.0_rk             ! mg/l
   real(rk) :: thickness_L1=0.2_rk       ! thickness of transport layer
   real(rk) :: thickness_fluff=0.001_rk  ! thickness of fluff layer
   real(rk) :: tauc_factor=10000.0_rk
   real(rk) :: M0=0.5e-2_rk              ! g/m^2/s
   real(rk) :: M1=3e-5_rk                ! 1/s
   real(rk) :: M2=3.5e-3_rk              ! g/m^2/s
```

```
   real(rk) :: flux_alpha=0.05          ! partition of deposition flux for
resuspension model 2
   real(rk) :: tauc_const=0.01_rk       ! N/m**2
   real(rk) :: ws_const=0.001_rk        ! m/s
   logical  :: cohesive=.false.
   logical  :: sand_mud_interaction=.false.
   integer  :: sinking_method=0
   integer  :: bedload_method=3
   integer  :: bottom_stress_method=1
   logical  :: pm_pool=.true.
   real(rk) :: shading=1.0_rk           ! 1/m per mg/l
   real(rk) :: rho=2650.0_rk            ! dry bed density kg/m**3
   logical  :: add_to_density=.false.   ! include density effects in EOS
   integer  :: resuspension_model=1     ! first order model, one layer
   real(rk) :: morfac=1.0_rk            ! morphological factor
   real(rk) :: bedload_factor=1.0_rk    ! morphological factor
   real(rk) :: crit_mud=0.5_rk          ! citical mud content
   real(rk) :: stressexponent=1.5_rk    ! flux = M[0|1]*(taub/
tauc-1)**stressexponent
                                        !   1.5 for sand and 1.0 for mud/cohesive

   logical  :: use_par=.false.

   namelist /iow_spm/  diameter, &
         c_init, tauc_factor, M0, M1, M2,                              &
         tauc_const, cohesive, sinking_method, bottom_stress_method,    &
         pm_pool, shading, ws_const, rho, bedload_method,               &
         add_to_density, sand_mud_interaction, crit_mud,stressexponent, &
         resuspension_model, thickness_L1, thickness_fluff, flux_alpha,    &
         morfac, bedload_factor, use_par
!EOP
!-----------------------------------------------------------------------
!BOC

   ! Read the namelist
   if (configunit>0) read(configunit,nml=iow_spm,err=99,end=100)

   ! Store parameter values in our own derived type
   call self%get_parameter(self%
diameter,'diameter','m',default=diameter,scale_factor=1e-6_rk)
   call self%get_parameter(self%tauc_factor ,'tauc_factor',default=tauc_factor)
   call self%get_parameter(self%tauc_const,'tauc_const',default=tauc_const)
   call self%get_parameter(self%cohesive,'cohesive',default=cohesive)
   call self%get_parameter(self%shading ,'shading',default=shading)
   call self%get_parameter(self%ws_const,'ws_const',default=ws_const)
   call self%get_parameter(self%
bottom_stress_method,'bottom_stress_method',default=bottom_stress_method)
   call self%get_parameter(self%
sinking_method,'sinking_method',default=sinking_method)
   call self%get_parameter(self%pm_pool,'pm_pool',default=pm_pool)
   call self%get_parameter(self%thickness_L1,'thickness_L1',default=thickness_L1)
   call self%get_parameter(self%
thickness_fluff,'thickness_fluff',default=thickness_fluff)
   call self%get_parameter(self%rho,'rho',default=rho)
   call self%get_parameter(self%M0,'M0',default=M0)
   call self%get_parameter(self%M1,'M1',default=M1)
   call self%get_parameter(self%M2,'M2',default=M2)
   call self%get_parameter(self%flux_alpha,'flux_alpha',default=flux_alpha)
   call self%get_parameter(self%
bedload_method,'bedload_method',default=bedload_method)
   call self%get_parameter(self%
bedload_factor,'bedload_factor',default=bedload_factor)
   call self%get_parameter(self%
add_to_density,'add_to_density',default=add_to_density)
   call self%get_parameter(self%
resuspension_model,'resuspension_model',default=resuspension_model)
```

```
   call self%get_parameter(self%morfac,'morfac',default=morfac)
   call self%get_parameter(self%
sand_mud_interaction,'sand_mud_interaction',default=sand_mud_interaction)
   call self%get_parameter(self%crit_mud,'crit_mud',default=crit_mud)
   call self%get_parameter(self%
stressexponent,'stressexponent',default=stressexponent)
   call self%get_parameter(self%use_par,'use_par',default=use_par)

   ! do some printing
   LEVEL2 ' particle diameter        : ',real(self%diameter)
   LEVEL2 ' cohesive                 : ',self%cohesive
   LEVEL2 ' bottom stress method     : ',self%bottom_stress_method
   LEVEL2 ' correct EOS              : ',self%add_to_density
   if ( self%morfac .ne. 1.0_rk ) then
      LEVEL2 ' Morphological factor     : ',real(self%morfac)
   endif
   select case (self%bottom_stress_method )
      case ( 0 )
         LEVEL2 ' constant critical bottom stress : ',real(self%tauc_const),' Pa'
      case ( 1 )
         LEVEL2 ' bottom stress method : Soulsby 1990 '
      case ( 2 )
         LEVEL2 ' bottom stress method : van Rijn 1984 '
   end select

   select case (self%sinking_method )
      case ( 0 )
         LEVEL2 ' sinking method : constant sinking speed : ',real(self%
ws_const),' m/s'
      case ( 1 )
         if ( self%cohesive ) then
            LEVEL2 ' sinking method : Krone 1963 '
         else
            LEVEL2 ' sinking method : Soulsby 1997 '
         endif
      case ( 2 )
         if ( self%cohesive ) then
            LEVEL2 ' sinking method : Winterwerp 2001 '
         else
            LEVEL2 ' sinking method : Stokes/Newton '
         endif
      case ( 3 )
         if ( self%cohesive ) then
            LEVEL2 ' sinking method : Mehta 1986 '
         else
            LEVEL2 ' sinking method : currently not defined'
         endif
   end select

   select case (self%resuspension_model )
      case ( 0 )
         LEVEL2 ' Zero order resuspension E=M0*(taub/taubc-1) '
      case ( 1 )
         LEVEL2 ' First order resuspension E=M1*mass_in_layer*(taub/taubc-1) '
      case ( 2 )
         LEVEL2 ' Fluff layer bed model (van Kessel et al. 2011) '
   end select

   ! bed load is only computed for non-cohesive spm
   if ( ( self%bedload_method .gt. 0 ) .and. self%cohesive ) then
      LEVEL2 'Bed load is only computed for non-cohesive spm!'
      self%bedload_method = 0
   endif

   select case ( self%bedload_method )
      case ( 0 )
```

```
          LEVEL2 ' bedload switched off '
      case ( 1 )
          LEVEL2 ' account for bedload : van Rijn 1984 '
      case ( 2 )
          LEVEL2 ' account for bedload : Nielsen 1992 '
      case ( 3 )
          LEVEL2 ' account for bedload : Engelund & Hansen 1972 '
      case ( 4 )
          LEVEL2 ' account for bedload : Lesser et al. 2004 '
   end select

   ! Register state variables
   call self%register_state_variable(self%id_spm,'spm','mg/l','concentration of
SPM',     &
                                    c_init,minimum=0.0_rk, &
                                    vertical_movement=self%ws_const, &
                                    no_river_dilution=.true.)

   if ( self%pm_pool )  then
      ! at first we need the bottom pool
      call self%register_state_variable(self%id_pmpool,'pmpool','kg/m**2', &
         'mass/m**2 of PM in bottom pool',self%rho*self%thickness_L1)
      ! compute the total mass in the bottom pool
      call self%register_dependency(self%
id_total_pmpool,'total_bedpool_at_interfaces')
      call self%add_to_aggregate_variable(type_bulk_standard_variable
(name='total_bedpool'), &
            self%id_pmpool)
      ! now figure out, which resuspension model is used
      select case (self%resuspension_model )
         case ( 0,1 )
            if ( self%bedload_method .gt. 0 ) then
               call self%register_horizontal_diagnostic_variable(self%id_bedload,
'bedload', &
                     'kg/m/s','massflux due to bed load')
            endif
         case ( 2 )
            ! if we use the van Kessel et al. 2011 formulation,
            ! we need at first a fluff layer ( with only mud )
            call self%register_state_variable(self%id_flufflayer,'flufflayer','kg/
m**2', &
               'mass/m**2 of sediment in fluff layer',self%rho*self%
thickness_fluff)
            ! compute the total mass in the fluff layer
            call self%register_dependency(self%
id_total_fluffpool,'total_fluffpool_at_interfaces')
            call self%add_to_aggregate_variable(type_bulk_standard_variable
(name='total_fluffpool'), &
               self%id_flufflayer)
            if ( self%sand_mud_interaction ) then
               ! compute the total mass of mud in the sand bed
               call self%add_to_aggregate_variable(type_bulk_standard_variable
(name='total_mudpool'), &
                  self%id_pmpool)
            endif
      end select
   endif

   ! diagnostic output of the massflux
   call self%register_horizontal_diagnostic_variable(self%
id_massflux,'massflux','kg/m**2/s', &
         'massflux in the exchange layer')
   ! and the mass fraction of each class
   call self%register_horizontal_diagnostic_variable(self%
id_massfraction,'massfraction','%', &
         'massfraction in the exchange layer')
```

```
   ! check for sand mud interaction
   if ( self%sand_mud_interaction ) then
      LEVEL2 ' Account for sand-mud interaction '
      ! check for valid range
      self%crit_mud = max(min(self%crit_mud,0.99_rk),0.0_rk)
      ! register link to the mudpool
      call self%register_dependency(self%
id_total_mudpool,'total_mudpool_at_interfaces')
      ! check if the diameter is larger than 63 mikron (limit for mud)
      if (self%diameter .gt. 63.0_rk/1e6_rk ) then
         LEVEL2 'This is the mud fraction'
         LEVEL2 'Limit the diameter to max 63 mikron!'
         self%diameter = 63.0_rk/1e6_rk
      endif
      bedload_method = 0
      LEVEL2 ' bedload switched off for mud'
   endif

   if ( self%use_par ) then
      call self%register_diagnostic_variable(self%id_dPAR,'PAR','W/
m**2','photosynthetically active radiation',   &
         time_treatment=time_treatment_averaged)
      call self%register_dependency(self%id_par, standard_variables%
downwelling_photosynthetic_radiative_flux)
   endif

   call self%set_variable_property(self%id_spm,'diameter',self%diameter)
   call self%set_variable_property(self%id_spm,'cohesive',self%cohesive)
   call self%set_variable_property(self%id_spm,'add_to_density',self%
add_to_density)
   call self%set_variable_property(self%id_spm,'density',self%rho)
   call self%set_variable_property(self%id_spm,'spm',.true.)
   call self%set_variable_property(self%id_spm,'morfac',self%morfac)

   if ( self%pm_pool ) then
      call self%set_variable_property(self%id_pmpool,'density',self%rho)
      call self%set_variable_property(self%id_pmpool,'morfac',self%morfac)
   end if

   ! Register environmental dependencies
   call self%register_dependency(self%id_taub, standard_variables%bottom_stress)
   call self%register_dependency(self%id_temp, standard_variables%temperature)
   call self%register_dependency(self%id_rhow, standard_variables%density)

   return

99 call self%fatal_error('iow_spm_create','Error reading namelist iow_spm')

100 call self%fatal_error('iow_spm_create','Namelist iow_spm was not found')

   end subroutine initialize
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Right hand sides of spm model
!
! !INTERFACE:
   subroutine do(self,_ARGUMENTS_DO_)
!!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
```

```
   class(type_iow_spm), INTENT(IN) :: self
  _DECLARE_ARGUMENTS_DO_
!
!
! !LOCAL VARIABLES:
    real(rk)        :: par
!EOP
!-----------------------------------------------------------------------
!BOC

   if ( self%use_par ) then

   !   Enter spatial_loops (if any)
       _LOOP_BEGIN_

   !   Retrieve current environmental conditions
   !    local photosynthetically active radiation
       _GET_(self%id_par,par)

   ! Export diagnostic variables
      _SET_DIAGNOSTIC_(self%id_dPAR,par)

   !   Leave spatial loops (if any)
      _LOOP_END_

   endif

   END subroutine do
!EOC

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

!-----------------------------------------------------------------------
!BOP
!
! !ROUTINE: Sedimentation/Erosion
!
! !INTERFACE:
   subroutine do_bottom(self,_ARGUMENTS_DO_BOTTOM_)
!
! !DESCRIPTION:
! Calculating the benthic fluxes
!
   implicit none

! !INPUT PARAMETERS:
   class (type_iow_spm), intent(in) :: self
   _DECLARE_ARGUMENTS_DO_BOTTOM_

! !LOCAL VARIABLES:
   real(rk)                     :: taub,spm,pmpool
   real(rk)                     :: porosity
   real(rk), parameter          :: rho_0=1025.0_rk
   real(rk)                     :: Erosion_Flux,Sedimentation_Flux
   real(rk)                     :: tauc_erosion, tauc_sedimentation
   real(rk)                     :: tauc_erosion_sandmud
   real(rk)                     :: tauc_erosion_fluff
   real(rk)                     :: temp
   real(rk)                     :: Ds
   real(rk)                     :: visc
   real(rk)                     :: theta
   real(rk)                     :: rhow
   real(rk)                     :: rhop
   real(rk)                     :: massflux
   real(rk)                     :: stressexponent=1.5_rk
   real(rk)                     :: qstar
```

```
   real(rk)                      :: bedload
   real(rk)                      :: mudfraction=0.0_rk
   real(rk)                      :: mudpool=0.0_rk
   real(rk)                      :: totalmass=0.0_rk
   real(rk)                      :: totalmass2=0.0_rk
   real(rk)                      :: E0, E1, E2, weight
   real(rk)                      :: stress

   real(rk)                      :: fluffraction,fluff,totalfluff
   real(rk)                      :: Erosion_Flux_fluff,Sedimentation_Flux_fluff
   real(rk)                      :: massflux_fluff

   real(rk), parameter           :: g=9.81_rk
!
! !REVISION HISTORY:
!  Original author(s): Richard Hofmeister & Ulf Gräwe
!
!EOP
!-----------------------------------------------------------------------
!BOC

   if ( self%pm_pool ) then
      ! we have to think about a proper porosity model !
      porosity=0.333_rk

      stressexponent = self%stressexponent

      _FABM_HORIZONTAL_LOOP_BEGIN_
      ! get environmental variables
      _GET_(self%id_spm,spm)
      _GET_(self%id_temp,temp)
      _GET_(self%id_rhow,rhow)
      _GET_HORIZONTAL_(self%id_taub,taub)
      ! get the total mass in the bottom
      _GET_HORIZONTAL_(self%id_total_pmpool,totalmass)
      ! get the mass of the individual class
      _GET_HORIZONTAL_(self%id_pmpool,pmpool)
      if ( self%resuspension_model .eq. 2 ) then
         ! do the same thing for the fluff layer
         _GET_HORIZONTAL_(self%id_flufflayer,fluff)
         _GET_HORIZONTAL_(self%id_total_fluffpool,totalfluff)
         fluffraction = fluff/totalfluff
      endif

      ! get the mass of the mud classes in the bottom pool
      if ( self%sand_mud_interaction ) then
         ! compute the mud fraction
         _GET_HORIZONTAL_(self%id_total_mudpool,mudpool)
         mudfraction = mudpool/totalmass
      endif

      select case ( self%bottom_stress_method )
         case ( 0 )
            tauc_erosion = self%tauc_const
         case ( 1 )
            ! Soulsby 1990
            rhop  = self%rho
            ! compute the dynamic viscosity
            visc  = 1.0_rk/rhow*1.9909e-6_rk*exp(1828.4_rk/(temp+273.15_rk))
            Ds    = (g/visc**2*(rhop/rhow-1.0_rk))**(0.3333_rk)*self%diameter
            theta = 0.3_rk/(1.0_rk + 1.2_rk*Ds) + 0.055_rk*(1.0_rk-exp
(-0.02_rk*Ds));
            tauc_erosion = g*self%diameter*(rhop-rhow)*theta
         case ( 2 )
            ! van Rijn 1984
            rhop  = self%rho
```

```
         visc  = 1.0_rk/rhow*1.9909e-6_rk*exp(1828.4_rk/(temp+273.15_rk))
         Ds    = (g/visc**2*(rhop/rhow-1.0_rk))**(0.3333_rk)*self%diameter
         if ( Ds .le. 4    ) then
            tauc_erosion = 0.24_rk*Ds**(-0.9_rk)
         elseif ( (Ds .gt. 4  ) .and. (Ds .le. 10 ) ) then
            tauc_erosion = 0.14_rk*Ds**(-0.64_rk)
         elseif ( (Ds .gt. 10 ) .and. (Ds .le. 20 ) ) then
            tauc_erosion = 0.04_rk*Ds**(-0.1_rk)
         elseif ( (Ds .gt. 20 ) .and. (Ds .le. 150) ) then
            tauc_erosion = 0.013_rk*Ds**(0.29_rk)
         else
            tauc_erosion = 0.056_rk
         endif
         tauc_erosion = tauc_erosion*g*self%diameter*(rhop-rhow)
   end select

   ! save the erosion stress for the fluff layer, since the fluff layer
   ! is not effected by the sand-mud interaction
   tauc_erosion_fluff = tauc_erosion
   if ( self%sand_mud_interaction ) then
      ! The change in increased critical shear should only act on the bottom
pool,
      ! not on the fluff layer!
      ! Van Rijn (1993) and Van Rijn (2004) non_cohesive regime
      if ( mudfraction .le. self%crit_mud ) then
         tauc_erosion = tauc_erosion*(1.0_rk+mudfraction)**2
      else
         tauc_erosion = tauc_erosion*(1.0_rk+self%crit_mud)**2
      endif
   endif

   tauc_sedimentation = tauc_erosion*self%tauc_factor

   Erosion_Flux             = 0.0_rk
   Erosion_Flux_fluff       = 0.0_rk
   Sedimentation_Flux       = 0.0_rk
   Sedimentation_Flux_fluff = 0.0_rk

   ! compute the excessive stress
   stress = max(taub/tauc_erosion-1.0_rk,0.0_rk)**stressexponent
   if ( self%sand_mud_interaction ) then
      if ( mudfraction .ge. self%crit_mud ) then
         ! stress is interpolated between
         !  the non-cohesive (sand) and fully mud regime
         weight = (1.0_rk - mudfraction)/(1.0_rk-self%crit_mud)
         stress = weight * max(taub/tauc_erosion-1.0_rk,0.0_rk)**stressexponent
+ &
                  (1.0_rk-weight) * max(taub/tauc_erosion_fluff-1.0_rk,0.0_rk)
      endif
   endif

   ! erosion flux:
   select case ( self%resuspension_model )
      case ( 0 )
         if( (pmpool .gt. 1.0_rk) .and. (taub .gt. tauc_erosion) ) then
            ! if there are sediments in the pool
            E0           = self%M0*(1.0_rk-porosity)
            Erosion_Flux = E0*stress
         endif
      case ( 1 )
         ! do a smooth transition between zero and first oder model, thus
         ! E ~ min(M0, pmpool*M1)
         E1           = min(self%M0,self%M1*pmpool)*(1.0_rk-porosity)
         Erosion_Flux = E1*stress
      case ( 2 )
         ! do a smooth transition between zero and first oder model
```

```
            Erosion_Flux_fluff = min(self%M0,self%M1*fluff*fluffraction) * &
                                    max(taub/tauc_erosion_fluff-1.0_rk,0.0_rk)
            E2                 = self%M2*fluffraction*(1.0_rk-porosity)
            Erosion_Flux       = E2*max
((taub/1.5_rk-1.0_rk),0.0_rk)**stressexponent
      end select

      !sedimentation flux:
      Sedimentation_Flux = min(0.0_rk,ws(self,temp,rhow,spm) * spm *(1.0_rk-taub /
tauc_sedimentation))
      if (self%resuspension_model .eq. 2 ) then
         ! 1-alpha deposits in the fluff layer, the remaining part
         ! goes into the bottom pool
         Sedimentation_Flux_fluff  = Sedimentation_Flux * (1.0_rk - self%
flux_alpha)
         Sedimentation_Flux        = Sedimentation_Flux * self%flux_alpha
      endif

      if ( self%bedload_method .gt. 0 ) then
         rhop  = self%rho
         visc  = 1.0_rk/rhow*1.9909e-6_rk*exp(1828.4_rk/(temp+273.15_rk))
         Ds    = (g/visc**2*(rhop/rhow-1.0_rk))**(0.3333_rk)*self%diameter
         select case ( self%bedload_method )
            case ( 1 )
               ! van Rijn 1984
               if( taub .gt. tauc_erosion ) then
                  qstar   = 0.053_rk*Ds**(-0.3_rk)*(taub/tauc_erosion-1.0_rk)**
(2.1)
                  bedload = qstar*sqrt(g*self%diameter**3*(rhop/rhow-1.0_rk))
               endif
            case ( 2 )
               ! Nielsen 1992
               if( taub .gt. tauc_erosion ) then
                  qstar   = 12.0_rk*(taub-tauc_erosion)/(g*self%diameter*(rhop-
rhow))* &
                            sqrt(taub/g*self%diameter*(rhop-rhow))
                  bedload = qstar*sqrt(g*self%diameter**3*(rhop/rhow-1.0_rk))
               endif
            case ( 3 )
               ! Engelund & Hansen 1972
               qstar   = 0.05_rk*(taub/(g*self%diameter*(rhop-rhow)))**2.5
               bedload = qstar*sqrt(g*self%diameter**3*(rhop/rhow-1.0_rk))
            case ( 4 )
               ! Lesser et al. 2004
               bedload = 0.5_rk*rhop*self%diameter*sqrt(taub/rhow)*Ds**(-0.3_rk)* &
                            taub/(g*self%diameter*(rhop-rhow))
               bedload = bedload*sqrt(g*self%diameter**2*(rhop/rhow-1.0_rk))
         end select
      endif

      ! compute mass flux
      select case ( self%resuspension_model )
         case ( 0,1 )
            massflux = Sedimentation_Flux+Erosion_Flux
            ! unit is g/m**3 * m/s
            _SET_BOTTOM_EXCHANGE_(self%id_spm,massflux)
            ! unit is kg/m**2/s
            _SET_ODE_BEN_(self%id_pmpool,-(massflux)/1000.0_rk*self%morfac)
            ! Export diagnostic variables mass flux - unit is kg/m**2/s
            _SET_HORIZONTAL_DIAGNOSTIC_(self%id_massflux,-(massflux)/1000.0_rk)
            ! Export diagnostic variables bed load flux - unit is kg/m/s
            if ( self%bedload_method .gt. 0 ) then
               _SET_HORIZONTAL_DIAGNOSTIC_(self%id_bedload,bedload*self%
bedload_factor*self%morfac)
            endif
         case ( 2 )
```

```
            massflux       = Sedimentation_Flux       + Erosion_Flux
            massflux_fluff = Sedimentation_Flux_fluff + Erosion_Flux_fluff
            ! unit is g/m**3 * m/s
            _SET_BOTTOM_EXCHANGE_(self%id_spm,(massflux+massflux_fluff))
            ! unit is kg/m**2/s
            _SET_ODE_BEN_(self%id_pmpool ,-(massflux)/1000.0_rk)
            _SET_ODE_BEN_(self%id_flufflayer,-(massflux_fluff)/1000.0_rk)
            ! Export diagnostic variables mass flux - unit is kg/m**2/s
            _SET_HORIZONTAL_DIAGNOSTIC_(self%id_massflux ,-(massflux )/1000.0_rk)
      end select

      _SET_HORIZONTAL_DIAGNOSTIC_(self%id_massfraction,pmpool/totalmass*100.0_rk)

      _FABM_HORIZONTAL_LOOP_END_

   end if

   end subroutine do_bottom
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !ROUTINE: iow_spm_get_vertical_movement
!
! !INTERFACE:
   subroutine get_vertical_movement(self,_FABM_ARGS_GET_VERTICAL_MOVEMENT_)

   implicit none

! !INPUT PARAMETERS:
   class (type_iow_spm), intent(in) :: self
   _DECLARE_FABM_ARGS_GET_VERTICAL_MOVEMENT_

! !LOCAL VARIABLES
   real(rk)      :: temp
   real(rk)      :: rhow
   real(rk)      :: spm
!
! !REVISION HISTORY:
!  Original author(s): Richard Hofmeister & Ulf Gräwe
!
   _FABM_LOOP_BEGIN_

   _GET_DEPENDENCY_(self%id_temp,temp)
   _GET_DEPENDENCY_(self%id_rhow,rhow)
   _GET_(self%id_spm,spm)
   _SET_VERTICAL_MOVEMENT_(self%id_spm,ws(self,temp,rhow,spm))

   _FABM_LOOP_END_

   end subroutine get_vertical_movement
!EOP
!-----------------------------------------------------------------------
!BOP
!
! !ROUTINE: iow_spm_get_light_extinction
!
! !INTERFACE:

   subroutine get_light_extinction(self,_ARGUMENTS_GET_EXTINCTION_)

   implicit none

! !INPUT PARAMETERS:
```

```fortran
   class (type_iow_spm), intent(in) :: self
   _DECLARE_ARGUMENTS_GET_EXTINCTION_

! !LOCAL VARIABLES
   real(rk)      :: spm
!
! !REVISION HISTORY:
!  Original author(s): Richard Hofmeister
!
!EOP
!-----------------------------------------------------------------------
!BOC
   ! Enter spatial loops (if any)
   _LOOP_BEGIN_

   _GET_(self%id_spm,spm)
   _SET_EXTINCTION_(self%shading*spm)

   _LOOP_END_

   end subroutine get_light_extinction
!EOC
!-----------------------------------------------------------------------

   function ws(self,temp,rhow,spm) result(svs)

   implicit none

   class (type_iow_spm), intent(in) :: self
   real(rk),             intent(in) :: temp
   real(rk),             intent(in) :: rhow     ! water density
   real(rk),             intent(in) :: spm      ! spm concentration

   real(rk)                         :: svs      ! sinking velocity in m/s

   real(rk)                         :: rhop     ! dry bed density of particle
   real(rk)                         :: visc     ! kinematic viscosity, visc=visc(T)
   real(rk)                         :: Ds       !
   real(rk)                         :: pRe      ! particle Reynolds number
   real(rk)                         :: Cd       ! dynamic drag coefficient
   real(rk)                         :: sF       ! shape factor
   real(rk)                         :: rrho     ! reduced density
   real(rk)                         :: phi      ! volumetric concentration
   real(rk)                         :: phistar  ! phi-limiter
   real(rk)                         :: a,b      ! parameter for cohesive sinking
velocity
   integer                          :: i

   real(rk), parameter              :: g=9.81_rk

   if ( self%cohesive ) then
      ! start with the cohesive sinking computation
      ! background sinking is based on Krone 1963
      ! spm must be in kg/m**3 and svs is in mm/s
      a   = 0.36_rk
      b   = 1.33_rk
      svs = (a*(spm/1000.0_rk)**b)/1000.0_rk

      select case ( self%sinking_method )
         case ( 0 )
            svs = abs(self%ws_const)
         case ( 1 )
            svs = svs
         case ( 2 )
            ! Winterwerp 2001
            rhop    = self%rho
```

```
            ! compute volumetric concentration
            phi     = min(0.9999999_rk,spm/g/rhop)
            ! do a limiter
            phistar = min(1.0_rk,phi)
            svs     = svs*(1.0_rk-phistar)*(1.0_rk-phi)/(1.0_rk+2.5_rk*phi)
         case ( 3 )
            ! Mehta 1986
            svs  = svs*(max(1.0_rk-0.008*spm/1000.0_rk,0.0_rk))**5

      end select
      svs = -svs
   else
      ! now do the non-cohesive sinking
      select case ( self%sinking_method )
         case ( 0 )
            svs = -abs(self%ws_const)
         case ( 1 )
            ! Soulsby R (1997) Dynamics of marine sands - a manual for practical
            !   applications. Thomas Telford, London
            rhop = self%rho
            visc = 1.0_rk/rhow*1.9909e-6_rk*exp(1.8284e3_rk/(temp+273.15_rk))
            Ds   = (g/visc**2*(rhop/rhow-1.0_rk))**(0.3333_rk)*self%diameter
            svs  = visc/self%diameter*(sqrt(10.36_rk**2+1.049_rk*Ds**3)-10.36_rk)
            svs = -svs

         case ( 2 )
            ! Stokes / Newton
            sF   = 0.64_rk ! assume spherical particles
            rhop = self%rho
            rrho = max((rhop-rhow)/rhow,0.0_rk)
            visc = 1.0_rk/rhow*1.9909e-6_rk*exp(1.8284e3_rk/(temp+273.15_rk))
            ! at first, estimate Stokes terminal sinking velocity
            svs  = self%diameter**2*g*rrho/18.0_rk/visc
            ! now, do the iteration since the drag depends on the sinking velocity
            ! UG: I think that 7 iterations are enough. One could also do a while
loop
            !     and test for the convergence
            do i=1,7
               pRe = sF*svs*self%diameter/visc
               ! The Cd formula is only valid for 1<pRe<1000, which is mostly the
case.
               Cd  = 18.5_rk/pRe**0.6_rk;
               svs = sqrt(4.0_rk*g*self%diameter*rrho/3.0_rk/Cd);
            enddo
            svs = -svs

      end select

   endif

   end function ws


   end module fabm_iow_spm
```