```
#include "cppdefs.h"
!-----------------------------------------------------------------------
!BOP
!
! !MODULE: input
!
! !INTERFACE:
   module input
!
! !DESCRIPTION:
!
! !USES:
   implicit none

!  default: all is private.
   private
!
! !PUBLIC MEMBER FUNCTIONS:
   public init_input, do_input, close_input, register_input
   public read_obs
   public type_input, type_scalar_input, type_profile_input
   public type_scalar_input_list

   integer, parameter, public :: method_unsupported = huge(1)
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!-----------------------------------------------------------------------
!

!  PRIVATE TYPES
   integer,parameter,public :: maxpathlen=256

   type type_input
      character(len=:), allocatable :: name
      integer                        :: method = 0     ! 0: constant, 2: from file
      REALTYPE                       :: scale_factor = _ONE_
      character(len=maxpathlen)      :: path = ''
      integer                        :: index = 1     ! Column index of variable in input file
      REALTYPE                       :: add_offset = _ZERO_
      REALTYPE                       :: constant_value = _ZERO_
      REALTYPE                       :: minimum = -huge(_ZERO_)
      REALTYPE                       :: maximum = huge(_ZERO_)

      integer                        :: method_off = method_unsupported
      integer                        :: method_constant = 0
      integer                        :: method_file = 2
   contains
      procedure :: configure
   end type

!  Information on an observed variable
   type, extends(type_input) :: type_profile_input
      REALTYPE, allocatable, dimension(:) :: data
   end type

   type, extends(type_input) :: type_scalar_input
      REALTYPE :: value = _ZERO_
   end type

   type type_scalar_input_node
      type (type_scalar_input),      pointer :: p => null()
      type (type_scalar_input_node), pointer :: next => null()
   end type

   type type_scalar_input_list
      type (type_scalar_input_node), pointer :: first => null()
   contains
      procedure :: add      => scalar_input_list_add
      procedure :: finalize => scalar_input_list_finalize
   end type

   type type_profile_input_node
      type (type_profile_input),      pointer :: p => null()
      type (type_profile_input_node), pointer :: next => null()
   end type

   type type_profile_input_list
      type (type_profile_input_node), pointer :: first => null()
   contains
      procedure :: add      => profile_input_list_add
      procedure :: finalize => profile_input_list_finalize
   end type

!  Information on file with observed profiles
   type type_profile_file
      character(len=maxpathlen)              :: path = ''
      REALTYPE, dimension(:,:), allocatable :: prof1,prof2,alpha
      integer                                :: jul1  = 0
      integer                                :: secs1 = 0
      integer                                :: jul2  = 0
      integer                                :: secs2 = 0
      integer                                :: unit  = -1
      integer                                :: lines = 0
      integer                                :: nprofiles = 0
      logical                                :: one_profile = .false.
      type (type_profile_input_list)        :: variables
      type (type_profile_file),pointer      :: next => null()
   contains
      procedure :: initialize => profile_file_initialize
      procedure :: update => profile_file_update
   end type

!  Information on file with observed scalars (time series)
   type type_timeseries_file
      character(len=maxpathlen)           :: path = ''
      REALTYPE, dimension(:), allocatable :: obs1,obs2,alpha
      integer                             :: jul1  = 0
      integer                             :: secs1 = 0
      integer                             :: jul2  = 0
      integer                             :: secs2 = 0
      integer                             :: unit = -1
      integer                             :: lines = 0
      integer                             :: n = 0
      type (type_scalar_input_list)       :: variables
      type (type_timeseries_file),pointer :: next => null()
   contains
      procedure :: initialize => timeseries_file_initialize
      procedure :: update => timeseries_file_update
   end type
!
!  PRIVATE PARAMETERS
   integer,parameter :: first_unit_no = 555

   type (type_scalar_input_list),  save :: scalar_inputs
   type (type_profile_input_list), save :: profile_inputs

!  PRIVATE DATA MEMBERS
!  Pointers to first files with observed profiles and observed scalars.
   type (type_profile_file),    pointer, save :: first_profile_file => null()
   type (type_timeseries_file), pointer, save :: first_timeseries_file => null()

!  Unit to use for next data file.
   integer, save :: next_unit_no = first_unit_no

   integer, save :: nlev = -1

   interface register_input
      module procedure register_scalar_input
      module procedure register_profile_input
   end interface

   contains

   subroutine configure(self, method, path, index, constant_value, scale_factor, add_offset, name)
      class (type_input),         intent(inout) :: self
      integer,          optional, intent(in)    :: method, index
      character(len=*), optional, intent(in)    :: path
      REALTYPE,         optional, intent(in)    :: constant_value, scale_factor, add_offset
      character(len=*), optional, intent(in)    :: name

      if (present(method)) self%method = method
      if (present(path)) self%path = path
      if (present(index) .and. self%method == self%method_file) self%index = index
      if (present(constant_value)) self%constant_value = constant_value
      if (present(scale_factor)) self%scale_factor = scale_factor
      if (present(add_offset)) self%add_offset = add_offset
      if (present(name)) self%name = name
   end subroutine

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Initialize input
!
! !INTERFACE:
   subroutine init_input(n)
!
! !DESCRIPTION:
!
! !INPUT PARAMETERS:
   integer,intent(in),optional :: n
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
!-----------------------------------------------------------------------
!BOC
   LEVEL1 'init_input'

   if (present(n)) then
      nlev = n
   else
      nlev = -1
   end if

   LEVEL2 'done'

   end subroutine init_input
!EOC

!-----------------------------------------------------------------------
```

```fortran
!BOP
!
! !IROUTINE: Register a 1d input variable.
!
! !INTERFACE:
   subroutine register_profile_input(input)
!
! !DESCRIPTION:
!
! !INPUT PARAMETERS:
   type (type_profile_input), target, intent(inout) :: input
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_profile_file), pointer :: file
!
!-----------------------------------------------------------------------
!BOC
   if (.not.allocated(input%name)) &
      call fatal_error('input::register_profile_input', 'input has not had a name assigned')

   if (nlev==-1) call fatal_error('input::register_profile_input', 'input module has not been initialized with depth information; &
      &depth-explicit inputs can therefore not be registered.')

   call profile_inputs%add(input)

   allocate(input%data(0:nlev))
   if (input%method == input%method_constant) then
      LEVEL2 'Using constant ' // input%name // '= ', input%constant_value
      input%data = input%constant_value
   elseif (input%method == input%method_file) then
      if (input%path=='') call fatal_error('input::register_profile_input', 'Empty file path specified to read variable '//input%name//' from.')

      LEVEL2 'Reading ' // input%name // ' from:'
      LEVEL3 trim(input%path)
      if (input%scale_factor /= 1) LEVEL3 'applying scale factor = ', input%scale_factor

      ! Find a file object for the specified file path; create one if it does exist yet.
      if (.not.associated(first_profile_file)) then
         allocate(first_profile_file)
         file => first_profile_file
      else
         file => first_profile_file
         do while (associated(file))
            if (file%path==input%path.and.file%unit==-1) exit
            file => file%next
         end do
         if (.not.associated(file)) then
            file => first_profile_file
            do while (associated(file%next))
               file => file%next
            end do
            allocate(file%next)
            file => file%next
         end if
      end if
      file%path = input%path
      call file%variables%add(input)
   else
      input%data = 0
   end if

   end subroutine register_profile_input
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Register a 0d input variable.
!
! !INTERFACE:
   subroutine register_scalar_input(input)
!
! !DESCRIPTION:
!
! !INPUT PARAMETERS:
   type (type_scalar_input), target, intent(inout) :: input
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_timeseries_file), pointer :: file
!
!-----------------------------------------------------------------------
!BOC
   if (.not.allocated(input%name)) &
      call fatal_error('input::register_scalar_input', 'input has not had a name assigned')

   call scalar_inputs%add(input)

   if (input%method == input%method_constant) then
      LEVEL2 'Using constant ' // input%name // '= ', input%constant_value
      input%value = input%constant_value
   elseif (input%method == input%method_file) then
      if (input%path=='') call fatal_error('input::register_scalar_input', 'Empty file path specified to read variabl
e '//input%name//' from.')

      LEVEL2 'Reading ' // input%name // ' from:'
      LEVEL3 trim(input%path)
      if (input%scale_factor /= 1) LEVEL3 'applying scale factor = ', input%scale_factor

      ! Find a file object for the specified file path; create one if it does exist yet.
      if (.not.associated(first_timeseries_file)) then
         allocate(first_timeseries_file)
         file => first_timeseries_file
      else
         file => first_timeseries_file
         do while (associated(file))
            if (file%path==input%path.and.file%unit==-1) exit
            file => file%next
         end do
         if (.not.associated(file)) then
            file => first_timeseries_file
            do while (associated(file%next))
               file => file%next
            end do
            allocate(file%next)
            file => file%next
         end if
      end if
      file%path = input%path
      call file%variables%add(input)
   else
      input%value = 0
   end if

   end subroutine register_scalar_input
!EOC

   subroutine scalar_input_list_add(self, input)
      class(type_scalar_input_list), intent(inout) :: self
      type(type_scalar_input), target              :: input

      type(type_scalar_input_node), pointer :: node

      if (associated(self%first)) then
         node => self%first
         do while (associated(node%next))
            node => node%next
         end do
         allocate(node%next)
         node => node%next
      else
         allocate(self%first)
         node => self%first
      end if
      node%p => input
   end subroutine

   subroutine profile_input_list_add(self, input)
      class(type_profile_input_list), intent(inout) :: self
      type(type_profile_input), target              :: input

      type(type_profile_input_node), pointer :: node

      if (associated(self%first)) then
         node => self%first
         do while (associated(node%next))
            node => node%next
         end do
         allocate(node%next)
         node => node%next
      else
         allocate(self%first)
         node => self%first
      end if
      node%p => input
   end subroutine

   subroutine scalar_input_list_finalize(self)
      class(type_scalar_input_list), intent(inout) :: self

      type(type_scalar_input_node), pointer :: node, next_node

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

   subroutine profile_input_list_finalize(self)
      class(type_profile_input_list), intent(inout) :: self

      type(type_profile_input_node), pointer :: node, next_node
```

```fortran

      node => self%first
      do while (associated(node))
         next_node => node%next
         if (allocated(node%p%data)) deallocate(node%p%data)
         deallocate(node)
         node => next_node
      end do
      self%first => null()
   end subroutine

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Read input observations
!
! !INTERFACE:
   subroutine do_input(jul,secs,nlev,z)
!
! !DESCRIPTION:
!  Read observations for all FABM variables for the current time.
!
! !USES:
!
! !INPUT PARAMETERS:
   integer,  intent(in)          :: jul,secs
   integer,  intent(in),optional :: nlev
   REALTYPE, intent(in),optional :: z(:)
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_profile_file),   pointer :: profile_file
   type (type_timeseries_file),pointer :: timeseries_file
!-----------------------------------------------------------------------
!BOC
   if (associated(first_profile_file) .and. .not. (present(nlev).and.present(z))) &
      call fatal_error('input::do_input', 'do_input must receive nlev and z since one or more depth-varying inputs ha
ve been registered.')

!  Loop over files with observed profiles.
   profile_file => first_profile_file
   do while (associated(profile_file))
      call profile_file%update(jul,secs,nlev,z)
      profile_file => profile_file%next
   end do

!  Loop over files with observed scalars.
   timeseries_file => first_timeseries_file
   do while (associated(timeseries_file))
      call timeseries_file%update(jul,secs)
      timeseries_file => timeseries_file%next
   end do

   end subroutine do_input
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Initialize a single input file with depth-explicit (1D) variables
!
! !INTERFACE:
   subroutine profile_file_initialize(self, nlev)
!
! !DESCRIPTION:
!  Initialize a single file with observed profiles.
!
! !USES:
!
! !INPUT PARAMETERS:
   class (type_profile_file), intent(inout) :: self
   integer,                   intent(in)    :: nlev
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_profile_input_node), pointer :: curvar
   integer :: nvar
   integer :: rc
   integer :: ios
!
!-----------------------------------------------------------------------
!BOC
!  Open the input file.
   open(next_unit_no,file=self%path,status='old',action='read',iostat=ios)
   if (ios /= 0) call fatal_error('input::profile_file_initialize', 'Unable to open "'//trim(self%path)//'" for readi
ng')

!  Opening was successful - store the file unit, and increment the next unit with 1.
   self%unit = next_unit_no
   next_unit_no = next_unit_no + 1
```

```

!  Determine the maximum number of columns that we need to read.
   nvar = 0
   curvar => self%variables%first
   do while (associated(curvar))
      nvar = max(nvar, curvar%p%index)
      curvar => curvar%next
   end do

   allocate(self%prof1(0:nlev,nvar),stat=rc)
   if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (prof1)'
   self%prof1 = 0

   allocate(self%prof2(0:nlev,nvar),stat=rc)
   if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (prof2)'
   self%prof2 = 0

   allocate(self%alpha(0:nlev,nvar),stat=rc)
   if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (alpha)'
   self%alpha = 0

   end subroutine profile_file_initialize
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Read 1D data from a single input file
!
! !INTERFACE:
   subroutine profile_file_update(self,jul,secs,nlev,z)
!
! !DESCRIPTION:
!  Get observations for the current time from a single input file.
!  This reads in new observations if necessary (and available),
!  and performs linear interpolation in time and vertical space.
!
! !USES:
   use time, only: time_diff,julian_day
!
! !INPUT PARAMETERS:
   integer,                  intent(in)   :: jul,secs
   integer,                  intent(in)   :: nlev
   REALTYPE,                 intent(in)   :: z(0:nlev)
!
! !INPUT/OUTPUT PARAMETERS:
   class(type_profile_file), intent(inout):: self
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   integer                        :: rc
   integer                        :: yy,mm,dd,hh,min,ss
   REALTYPE                       :: t,dt
   type (type_profile_input_node), pointer :: curvar
   character(len=128)             :: strline
!
!-----------------------------------------------------------------------
!BOC
   if (self%unit==-1) call self%initialize(nlev)

   if (self%one_profile) return

!  This part reads in new values if necessary.
   if(time_diff(self%jul2,self%secs2,jul,secs)<0) then
      do
         self%jul1 = self%jul2
         self%secs1 = self%secs2
         self%prof1 = self%prof2
         call read_profiles(self%unit,nlev,ubound(self%prof2,2),yy,mm,dd,hh,min,ss,z,self%prof2,self%lines,rc)
         if(rc/=0) then
            if (rc<0) then
               if(self%nprofiles==1) then
                  LEVEL3 'Only one set of profiles is present in '//trim(self%path)//'. These will be used throughout
 the simulation'
                  self%one_profile = .true.
                  curvar => self%variables%first
                  do while (associated(curvar))
                     curvar%p%data = self%prof1(:,curvar%p%index)
                     curvar => curvar%next
                  end do
               else
                  call fatal_error('input::profile_file_update', 'End of file reached while attempting to read new da
ta from '//trim(self%path)//'. Does this file span the entire simulated period?')
               end if
            else
               write (strline,'(i0)') self%lines
               call fatal_error('input::profile_file_update', 'Error reading profiles from '//trim(self%path)//' at l
ine '//trim(strline))
            end if
            return
         end if

         ! Apply offsets and scale factors to newly read profile
         curvar => self%variables%first
```

```fortran
         do while (associated(curvar))
            self%prof2(:,curvar%p%index) = curvar%p%scale_factor * self%prof2(:,curvar%p%index) + curvar%p%add_offset
            if (any(self%prof2(:,curvar%p%index) < curvar%p%minimum)) then
               write (strline,'(a,a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
                  'One or more values of the ',trim(curvar%p%name),' profile at ',yy,mm,dd,hh,min,ss, &
                  ' lie below prescribed minimum of ',curvar%p%minimum,'.'
               call fatal_error('input::profile_file_update', trim(self%path)//': '//trim(strline))
            end if
            if (any(self%prof2(:,curvar%p%index) > curvar%p%maximum)) then
               write (strline,'(a,a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
                  'One or more values of the ',trim(curvar%p%name),' profile at ',yy,mm,dd,hh,min,ss, &
                  ' exceed the prescribed maximum of ',curvar%p%maximum,'.'
               call fatal_error('input::profile_file_update', trim(self%path)//': '//trim(strline))
            end if
            curvar => curvar%next
         end do

         self%nprofiles = self%nprofiles + 1
         call julian_day(yy,mm,dd,self%jul2)
         self%secs2 = hh*3600 + min*60 + ss
         if(time_diff(self%jul2,self%secs2,jul,secs) > 0) exit
      end do
      if (self%nprofiles == 1) call fatal_error('input::profile_file_update', 'Simulation starts before time of first observation in '//trim(self%path)//'.')

      ! Compute slopes (change in variable per second)
      dt = time_diff(self%jul2,self%secs2,self%jul1,self%secs1)
      self%alpha = (self%prof2-self%prof1)/dt
   end if

   ! Perform time interpolation
   t = time_diff(jul,secs,self%jul1,self%secs1)
   curvar => self%variables%first
   do while (associated(curvar))
      curvar%p%data = self%prof1(:,curvar%p%index) + t * self%alpha(:,curvar%p%index)
      curvar => curvar%next
   end do

   end subroutine profile_file_update
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Initialize a single input file with horizontal (0D) variables.
!
! !INTERFACE:
   subroutine timeseries_file_initialize(self)
!
! !DESCRIPTION:
!  Initialize a single file with observed profiles.
!
! !INPUT PARAMETERS:
   class (type_timeseries_file),intent(inout) :: self
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_scalar_input_node),pointer :: curvar
   integer :: nvar
   integer :: rc
   integer :: ios
!
!-----------------------------------------------------------------------
!BOC
!  Open the input file.
   open(next_unit_no,file=self%path,status='old',action='read',iostat=ios)
   if (ios /= 0) call fatal_error('input::timeseries_file_initialize', 'Unable to open "'//trim(self%path)//'" for reading')

!  Opening was successful - store the file unit, and increment the next unit with 1.
   self%unit = next_unit_no
   next_unit_no = next_unit_no + 1

!  Determine the maximum number of columns that we need to read.
   nvar = 0
   curvar => self%variables%first
   do while (associated(curvar))
      nvar = max(nvar,curvar%p%index)
      curvar => curvar%next
   end do

   allocate(self%obs1(nvar),stat=rc)
   if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (obs1)'
   self%obs1 = 0

   allocate(self%obs2(nvar),stat=rc)
   if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (obs2)'
   self%obs2 = 0

   allocate(self%alpha(nvar),stat=rc)
   if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (alpha)'
   self%alpha = 0

   end subroutine timeseries_file_initialize
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Read 0D data from a single input file
!
! !INTERFACE:
   subroutine timeseries_file_update(self,jul,secs)
!
! !DESCRIPTION:
!  Get observations for the current time from a single input file.
!  This reads in new observations if necessary (and available),
!  and performs linear interpolation in time.
!
! !USES:
   use time, only: time_diff,julian_day
!
! !INPUT PARAMETERS:
   integer,                    intent(in)    :: jul,secs
!
! !INPUT/OUTPUT PARAMETERS:
   class(type_timeseries_file), intent(inout) :: self
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   integer                      :: rc
   integer                      :: yy,mm,dd,hh,mins,ss
   REALTYPE                     :: t,dt
   type (type_scalar_input_node),pointer :: curvar
   character(len=128)           :: strline
!
!-----------------------------------------------------------------------
!BOC
   if (self%unit==-1) call self%initialize()

!  This part reads in new values if necessary.
   if(time_diff(self%jul2,self%secs2,jul,secs) < 0) then
      do
         self%jul1 = self%jul2
         self%secs1 = self%secs2
         self%obs1 = self%obs2
         call read_obs(self%unit,yy,mm,dd,hh,mins,ss,size(self%obs2),self%obs2,rc,line=self%lines)
         if (rc>0) then
            write (strline,'(i0)') self%lines
            call fatal_error('input::timeseries_file_update', 'Error reading time series from '//trim(self%path)//' at line '//strline)
         elseif (rc<0) then
            call fatal_error('input::timeseries_file_update', 'End of file reached while attempting to read new data from '//trim(self%path)//'. Does this file span the entire simulated period?')
         end if

         ! Apply offsets and scale factors to newly read data
         curvar => self%variables%first
         do while (associated(curvar))
            self%obs2(curvar%p%index) = curvar%p%scale_factor * self%obs2(curvar%p%index) + curvar%p%add_offset
            if (self%obs2(curvar%p%index) < curvar%p%minimum) then
               write (strline,'(a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
                  trim(curvar%p%name),' at ',yy,mm,dd,hh,mins,ss, &
                  ' lies below prescribed minimum of ',curvar%p%minimum,'.'
               call fatal_error('input::timeseries_file_update', trim(self%path)//': '//trim(strline))
            end if
            if (self%obs2(curvar%p%index) > curvar%p%maximum) then
               write (strline,'(a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
                  trim(curvar%p%name),' at ',yy,mm,dd,hh,mins,ss, &
                  ' exceeds the prescribed maximum of ',curvar%p%maximum,'.'
               call fatal_error('input::timeseries_file_update', trim(self%path)//': '//trim(strline))
            end if
            curvar => curvar%next
         end do

         self%n = self%n + 1
         call julian_day(yy,mm,dd,self%jul2)
         self%secs2 = hh*3600 + mins*60 + ss
         if(time_diff(self%jul2,self%secs2,jul,secs) > 0) exit
      end do
      if (self%n == 1) call fatal_error('input::timeseries_file_update', 'Simulation starts before time of first observation in '//trim(self%path)//'.')

      ! Compute slopes (change in variable per second)
      dt = time_diff(self%jul2,self%secs2,self%jul1,self%secs1)
      self%alpha = (self%obs2 - self%obs1) / dt
   end if

   ! Perform time interpolation
   t = time_diff(jul,secs,self%jul1,self%secs1)
   curvar => self%variables%first
   do while (associated(curvar))
      curvar%p%value = min(max(self%obs1(curvar%p%index), self%obs2(curvar%p%index)), max(min(self%obs1(curvar%p%index), self%obs2(curvar%p%index)), self%obs1(curvar%p%index) + t * self%alpha(curvar%p%index)))
      curvar => curvar%next
   end do

   end subroutine timeseries_file_update
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Close inputs
!
! !INTERFACE:
   subroutine close_input()
!
! !DESCRIPTION:
!
! !INPUT PARAMETERS:
!
! !REVISION HISTORY:
!  Original author(s): Jorn Bruggeman
!
!EOP
!
! !LOCAL VARIABLES:
   type (type_profile_file),     pointer :: profile_file,next_profile_file
   type (type_timeseries_file),  pointer :: timeseries_file,next_scalar_file
   type (type_profile_input_node),pointer :: curvar_1d,nextvar_1d
   type (type_scalar_input_node),pointer :: curvar_0d,nextvar_0d
!
!-----------------------------------------------------------------------
!BOC

   profile_file => first_profile_file
   do while (associated(profile_file))
      call profile_file%variables%finalize()

      next_profile_file => profile_file%next
      if (profile_file%unit/=-1) close(profile_file%unit)
      if (allocated(profile_file%prof1)) deallocate(profile_file%prof1)
      if (allocated(profile_file%prof2)) deallocate(profile_file%prof2)
      if (allocated(profile_file%alpha)) deallocate(profile_file%alpha)
      deallocate(profile_file)

      profile_file => next_profile_file
   end do
   nullify(first_profile_file)

   timeseries_file => first_timeseries_file
   do while (associated(timeseries_file))
      call timeseries_file%variables%finalize()

      next_scalar_file => timeseries_file%next
      if (timeseries_file%unit/=-1) close(timeseries_file%unit)
      if (allocated(timeseries_file%obs1))  deallocate(timeseries_file%obs1)
      if (allocated(timeseries_file%obs2))  deallocate(timeseries_file%obs2)
      if (allocated(timeseries_file%alpha)) deallocate(timeseries_file%alpha)
      deallocate(timeseries_file)

      timeseries_file => next_scalar_file
   end do
   nullify(first_timeseries_file)

   call scalar_inputs%finalize()
   call profile_inputs%finalize()

   next_unit_no = first_unit_no
   nlev = -1

   end subroutine close_input
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: read_obs
!
! !INTERFACE:
   subroutine read_obs(unit,yy,mm,dd,hh,min,ss,N,obs,ios,line)
!
! !DESCRIPTION:
!  This routine will read all non-profile observations.
!  The routine allows for reading more than one scalar variable at a time.
!  The number of data to be read is specified by {\tt N}.
!  Data read-in are returned
!  in the 'obs' array. It is up to the calling routine to assign
!  meaning full variables to the individual elements in {\tt obs}.
!
! !INPUT PARAMETERS:
   integer, intent(in)                 :: unit
   integer, intent(in)                 :: N
!
! !OUTPUT PARAMETERS:
   integer, intent(out)                :: yy,mm,dd,hh,min,ss
   REALTYPE,intent(out)                :: obs(:)
   integer, intent(out)                :: ios
   integer, intent(inout), optional    :: line
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
```

```fortran
!
! !LOCAL VARIABLES:
   integer                   :: i
   character                 :: c1,c2,c3,c4
   character(len=128)        :: cbuf
!-----------------------------------------------------------------------
!BOC
   do
      if (present(line)) line = line + 1
      read(unit,'(A128)',iostat=ios) cbuf
      if (ios/=0) return
      if (cbuf(1:1)/='#' .and. cbuf(1:1)/='!' .and. len_trim(cbuf)/=0) then
         read(cbuf,'(i4,a1,i2,a1,i2,1x,i2,a1,i2,a1,i2)',iostat=ios) yy,c1,mm,c2,dd,hh,c3,min,c4,ss
         if (ios==0) read(cbuf(20:),*,iostat=ios) (obs(i),i=1,N)
         if (ios<0) ios = 1   ! End-of-file (ios<0) means premature end of line, which is a read error (ios>0) to us
         return
      end if
   end do
   end subroutine read_obs
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: read_profiles
!
! !INTERFACE:
   subroutine read_profiles(unit,nlev,cols,yy,mm,dd,hh,min,ss,z, &
                            profiles,lines,ios)
!
! !DESCRIPTION:
!  Similar to {\tt read\_obs()} but used for reading profiles instead of
!  scalar data.
!  The data will be interpolated on the grid specified by nlev and z.
!  The data can be read 'from the top' or 'from the bottom' depending on
!  a flag in the actual file.
!
! !INPUT PARAMETERS:
   integer, intent(in)                  :: unit
   integer, intent(in)                  :: nlev,cols
   REALTYPE, intent(in)                 :: z(:)
!
! !INPUT/OUTPUT PARAMETERS:
   integer, intent(inout)               :: lines
!
! !OUTPUT PARAMETERS:
   integer, intent(out)                 :: yy,mm,dd,hh,min,ss
   REALTYPE, intent(out)                :: profiles(:,:)
   integer, intent(out)                 :: ios
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
!
! !LOCAL VARIABLES:
   character                 :: c1,c2,c3,c4
   integer                   :: i,j,rc
   integer                   :: N,up_down
   REALTYPE,allocatable,dimension(:)   :: tmp_depth
   REALTYPE,allocatable,dimension(:,:) :: tmp_profs
   character(len=128)        :: cbuf
   integer                   :: idx1,idx2,stride
!-----------------------------------------------------------------------
!BOC
   do
      read(unit,'(A128)', iostat=ios) cbuf
      lines = lines + 1
      if (ios /= 0) return

      if (len_trim(cbuf) > 0 .and. .not.(cbuf(1:1) == '#' .or. cbuf(1:1) == '!')) then
         read(cbuf,'(i4,a1,i2,a1,i2,1x,i2,a1,i2,a1,i2)',iostat=ios) yy,c1,mm,c2,dd,hh,c3,min,c4,ss
         if (ios < 0) ios = 1   ! End-of-file (ios<0) means premature end of line, which is a read error (ios>0) to u
s
         if (ios /= 0) return
         read(cbuf(20:),*,iostat=ios) N,up_down
         if (ios < 0) ios = 1   ! End-of-file (ios<0) means premature end of line, which is a read error (ios>0) to u
s
         if (ios /= 0) return
         exit
      end if
   end do

   allocate(tmp_depth(0:N),stat=rc)
   if (rc /= 0) stop 'read_profiles: Error allocating memory (tmp_depth)'
   allocate(tmp_profs(0:N,cols),stat=rc)
   if (rc /= 0) stop 'read_profiles: Error allocating memory (tmp_profs)'

   if(up_down .eq. 1) then
      idx1=1; idx2 = N; stride=1
   else
      idx1=N; idx2 = 1; stride=-1
   end if

   do i=idx1,idx2,stride
      do
         read(unit,'(A128)',iostat=ios) cbuf
         lines = lines + 1
         if (ios /= 0) return

         if (len_trim(cbuf) > 0 .and. .not. (cbuf(1:1) == '#' .or. cbuf(1:1) == '!')) then
            read(cbuf,*,iostat=ios) tmp_depth(i),(tmp_profs(i,j),j=1,cols)
            if (ios < 0) ios = 1   ! End-of-file (ios<0) means premature end of line, which is a read error (ios>0) to us
            if (ios /= 0) return
            exit
         end if
      end do
   end do

   call gridinterpol(N,cols,tmp_depth,tmp_profs,nlev,z,profiles)

   deallocate(tmp_depth)
   deallocate(tmp_profs)

   end subroutine read_profiles
!EOC

   subroutine fatal_error(location,error)
      character(len=*),  intent(in) :: location,error

      FATAL trim(location)//': '//trim(error)
      stop 1
   end subroutine fatal_error

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

   end module input

!-----------------------------------------------------------------------
! Copyright by the GOTM-team under the GNU Public License - www.gnu.org
!----------------------------------------------------------------------
```