```
#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

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

```
197 !BOP
198 !
199 ! !IROUTINE: Register a 1d input variable.
200 !
201 ! !INTERFACE:
202    subroutine register_profile_input(input)
203 !
204 ! !ĐESCRIPTION:
205 !
206 ! !INPUT PARAMETERS:
207    type (type_profile_input), target, intent(inout) :: input
208 !
209 ! !REVISION HISTORY:
210 !  Original author(s): Jorn Bruggeman
211 !
212 !EOP
213 !
214 ! !LOCAL VARIABLES:
215    type (type_profile_file), pointer :: file
216 !
217 !-----------------------------------------------------------------------
218 !BOC
219    if (.not.allocated(input%name)) &
220       call fatal_error('input::register_profile_input', 'input has not had a name assigned')
221
222    if (nlev==-1) call fatal_error('input::register_profile_input', 'input module has not been initialized with depth
    information; &
223       &depth-explicit inputs can therefore not be registered.')
224
225    call profile_inputs%add(input)
226
227    allocate(input%data(0:nlev))
228    if (input%method == input%method_constant) then
229       LEVEL2 'Using constant ' // input%name // '= ', input%constant_value
230       input%data = input%constant_value
231    elseif (input%method == input%method_file) then
232       if (input%path=='') call fatal_error('input::register_profile_input', 'Empty file path specified to read variab
    le '//input%name//' from.')
233
234       LEVEL2 'Reading ' // input%name // ' from:'
235       LEVEL3 trim(input%path)
236       if (input%scale_factor /= 1) LEVEL3 'applying scale factor = ', input%scale_factor
237
238       ! Find a file object for the specified file path; create one if it does exist yet.
239       if (.not.associated(first_profile_file)) then
240          allocate(first_profile_file)
241          file => first_profile_file
242       else
243          file => first_profile_file
244          do while (associated(file))
245             if (file%path==input%path.and.file%unit==-1) exit
246             file => file%next
247          end do
248          if (.not.associated(file)) then
249             file => first_profile_file
250             do while (associated(file%next))
251                file => file%next
252             end do
253             allocate(file%next)
254             file => file%next
255          end if
256       end if
257       file%path = input%path
258       call file%variables%add(input)
259    else
260       input%data = 0
261    end if
262
263    end subroutine register_profile_input
264 !EOC
265
266 !-----------------------------------------------------------------------
267 !BOP
268 !
269 ! !IROUTINE: Register a 0d input variable.
270 !
271 ! !INTERFACE:
272    subroutine register_scalar_input(input)
273 !
274 ! !ĐESCRIPTION:
275 !
276 ! !INPUT PARAMETERS:
277    type (type_scalar_input), target, intent(inout) :: input
278 !
279 ! !REVISION HISTORY:
280 !  Original author(s): Jorn Bruggeman
281 !
282 !EOP
283 !
284 ! !LOCAL VARIABLES:
285    type (type_timeseries_file), pointer :: file
286 !
287 !-----------------------------------------------------------------------
288 !BOC
289    if (.not.allocated(input%name)) &
290       call fatal_error('input::register_scalar_input', 'input has not had a name assigned')
291
292    call scalar_inputs%add(input)
```

```
293
294    if (input%method == input%method_constant) then
295       LEVEL2 'Using constant ' // input%name // '= ', input%constant_value
296       input%value = input%constant_value
297    elseif (input%method == input%method_file) then
298       if (input%path=='') call fatal_error('input::register_scalar_input', 'Empty file path specified to read variabl
   e '//input%name//' from.')
299
300       LEVEL2 'Reading ' // input%name // ' from:'
301       LEVEL3 trim(input%path)
302       if (input%scale_factor /= 1) LEVEL3 'applying scale factor = ', input%scale_factor
303
304       ! Find a file object for the specified file path; create one if it does exist yet.
305       if (.not.associated(first_timeseries_file)) then
306          allocate(first_timeseries_file)
307          file => first_timeseries_file
308       else
309          file => first_timeseries_file
310          do while (associated(file))
311             if (file%path==input%path.and.file%unit==-1) exit
312             file => file%next
313          end do
314          if (.not.associated(file)) then
315             file => first_timeseries_file
316             do while (associated(file%next))
317                file => file%next
318             end do
319             allocate(file%next)
320             file => file%next
321          end if
322       end if
323       file%path = input%path
324       call file%variables%add(input)
325    else
326       input%value = 0
327    end if
328
329    end subroutine register_scalar_input
330 !EOC
331
332    subroutine scalar_input_list_add(self, input)
333       class(type_scalar_input_list), intent(inout) :: self
334       type(type_scalar_input), target              :: input
335
336       type(type_scalar_input_node), pointer :: node
337
338       if (associated(self%first)) then
339          node => self%first
340          do while (associated(node%next))
341             node => node%next
342          end do
343          allocate(node%next)
344          node => node%next
345       else
346          allocate(self%first)
347          node => self%first
348       end if
349       node%p => input
350    end subroutine
351
352    subroutine profile_input_list_add(self, input)
353       class(type_profile_input_list), intent(inout) :: self
354       type(type_profile_input), target              :: input
355
356       type(type_profile_input_node), pointer :: node
357
358       if (associated(self%first)) then
359          node => self%first
360          do while (associated(node%next))
361             node => node%next
362          end do
363          allocate(node%next)
364          node => node%next
365       else
366          allocate(self%first)
367          node => self%first
368       end if
369       node%p => input
370    end subroutine
371
372    subroutine scalar_input_list_finalize(self)
373       class(type_scalar_input_list), intent(inout) :: self
374
375       type(type_scalar_input_node), pointer :: node, next_node
376
377       node => self%first
378       do while (associated(node))
379          next_node => node%next
380          deallocate(node)
381          node => next_node
382       end do
383       self%first => null()
384    end subroutine
385
386    subroutine profile_input_list_finalize(self)
387       class(type_profile_input_list), intent(inout) :: self
388
389       type(type_profile_input_node), pointer :: node, next_node
```

```
390
391       node => self%first
392       do while (associated(node))
393          next_node => node%next
394          if (allocated(node%p%data)) deallocate(node%p%data)
395          deallocate(node)
396          node => next_node
397       end do
398       self%first => null()
399    end subroutine
400
401 !-----------------------------------------------------------------------
402 !BOP
403 !
404 ! !IROUTINE: Read input observations
405 !
406 ! !INTERFACE:
407    subroutine do_input(jul,secs,nlev,z)
408 !
409 ! !DESCRIPTION:
410 !  Read observations for all FABM variables for the current time.
411 !
412 ! !USES:
413 !
414 ! !INPUT PARAMETERS:
415    integer,  intent(in)          :: jul,secs
416    integer,  intent(in),optional :: nlev
417    REALTYPE, intent(in),optional :: z(:)
418 !
419 ! !REVISION HISTORY:
420 !  Original author(s): Jorn Bruggeman
421 !
422 !EOP
423 !
424 ! !LOCAL VARIABLES:
425    type (type_profile_file),   pointer :: profile_file
426    type (type_timeseries_file),pointer :: timeseries_file
427 !-----------------------------------------------------------------------
428 !BOC
429    if (associated(first_profile_file) .and. .not. (present(nlev).and.present(z))) &
430       call fatal_error('input::do_input', 'do_input must receive nlev and z since one or more depth-varying inputs ha
    ve been registered.')
431
432 !  Loop over files with observed profiles.
433    profile_file => first_profile_file
434    do while (associated(profile_file))
435       call profile_file%update(jul,secs,nlev,z)
436       profile_file => profile_file%next
437    end do
438
439 !  Loop over files with observed scalars.
440    timeseries_file => first_timeseries_file
441    do while (associated(timeseries_file))
442       call timeseries_file%update(jul,secs)
443       timeseries_file => timeseries_file%next
444    end do
445
446    end subroutine do_input
447 !EOC
448
449 !-----------------------------------------------------------------------
450 !BOP
451 !
452 ! !IROUTINE: Initialize a single input file with depth-explicit (1D) variables
453 !
454 ! !INTERFACE:
455    subroutine profile_file_initialize(self, nlev)
456 !
457 ! !DESCRIPTION:
458 !  Initialize a single file with observed profiles.
459 !
460 ! !USES:
461 !
462 ! !INPUT PARAMETERS:
463    class (type_profile_file), intent(inout) :: self
464    integer,                   intent(in)    :: nlev
465 !
466 ! !REVISION HISTORY:
467 !  Original author(s): Jorn Bruggeman
468 !
469 !EOP
470 !
471 ! !LOCAL VARIABLES:
472    type (type_profile_input_node), pointer :: curvar
473    integer :: nvar
474    integer :: rc
475    integer :: ios
476 !
477 !-----------------------------------------------------------------------
478 !BOC
479 !  Open the input file.
480    open(next_unit_no,file=self%path,status='old',action='read',iostat=ios)
481    if (ios /= 0) call fatal_error('input::profile_file_initialize', 'Unable to open "'//trim(self%path)//'" for readi
    ng')
482
483 !  Opening was successful - store the file unit, and increment the next unit with 1.
484    self%unit = next_unit_no
485    next_unit_no = next_unit_no + 1
```

```
486
487 !  Determine the maximum number of columns that we need to read.
488    nvar = 0
489    curvar => self%variables%first
490    do while (associated(curvar))
491       nvar = max(nvar, curvar%p%index)
492       curvar => curvar%next
493    end do
494
495    allocate(self%prof1(0:nlev,nvar),stat=rc)
496    if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (prof1)'
497    self%prof1 = 0
498
499    allocate(self%prof2(0:nlev,nvar),stat=rc)
500    if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (prof2)'
501    self%prof2 = 0
502
503    allocate(self%alpha(0:nlev,nvar),stat=rc)
504    if (rc /= 0) stop 'input::profile_file_initialize: Error allocating memory (alpha)'
505    self%alpha = 0
506
507    end subroutine profile_file_initialize
508 !EOC
509
510 !-----------------------------------------------------------------------
511 !BOP
512 !
513 ! !IROUTINE: Read 1D data from a single input file
514 !
515 ! !INTERFACE:
516    subroutine profile_file_update(self,jul,secs,nlev,z)
517 !
518 ! !DESCRIPTION:
519 !  Get observations for the current time from a single input file.
520 !  This reads in new observations if necessary (and available),
521 !  and performs linear interpolation in time and vertical space.
522 !
523 ! !USES:
524    use time, only: time_diff,julian_day
525 !
526 ! !INPUT PARAMETERS:
527    integer,                 intent(in)   :: jul,secs
528    integer,                 intent(in)   :: nlev
529    REALTYPE,                intent(in)   :: z(0:nlev)
530 !
531 ! !INPUT/OUTPUT PARAMETERS:
532    class(type_profile_file), intent(inout):: self
533 !
534 ! !REVISION HISTORY:
535 !  Original author(s): Jorn Bruggeman
536 !
537 !EOP
538 !
539 ! !LOCAL VARIABLES:
540    integer                      :: rc
541    integer                      :: yy,mm,dd,hh,min,ss
542    REALTYPE                     :: t,dt
543    type (type_profile_input_node), pointer :: curvar
544    character(len=128)           :: strline
545 !
546 !-----------------------------------------------------------------------
547 !BOC
548    if (self%unit==-1) call self%initialize(nlev)
549
550    if (self%one_profile) return
551
552 !  This part reads in new values if necessary.
553    if(time_diff(self%jul2,self%secs2,jul,secs)<0) then
554       do
555          self%jul1 = self%jul2
556          self%secs1 = self%secs2
557          self%prof1 = self%prof2
558          call read_profiles(self%unit,nlev,ubound(self%prof2,2),yy,mm,dd,hh,min,ss,z,self%prof2,self%lines,rc)
559          if(rc/=0) then
560             if (rc<0) then
561                if(self%nprofiles==1) then
562                   LEVEL3 'Only one set of profiles is present in '//trim(self%path)//'. These will be used throughout
     the simulation'
563                   self%one_profile = .true.
564                   curvar => self%variables%first
565                   do while (associated(curvar))
566                      curvar%p%data = self%prof1(:,curvar%p%index)
567                      curvar => curvar%next
568                   end do
569                else
570                   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?')
571                end if
572             else
573                write (strline,'(i0)') self%lines
574                call fatal_error('input::profile_file_update', 'Error reading profiles from '//trim(self%path)//' at l
    ine '//trim(strline))
575             end if
576             return
577          end if
578
579          ! Apply offsets and scale factors to newly read profile
580          curvar => self%variables%first
```

```
581          do while (associated(curvar))
582             self%prof2(:,curvar%p%index) = curvar%p%scale_factor * self%prof2(:,curvar%p%index) + curvar%p%add_offset
583             if (any(self%prof2(:,curvar%p%index) < curvar%p%minimum)) then
584                write (strline,'(a,a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
585                   'One or more values of the ',trim(curvar%p%name),' profile at ',yy,mm,dd,hh,min,ss, &
586                   ' lie below prescribed minimum of ',curvar%p%minimum,'.'
587                call fatal_error('input::profile_file_update', trim(self%path)//': '//trim(strline))
588             end if
589             if (any(self%prof2(:,curvar%p%index) > curvar%p%maximum)) then
590                write (strline,'(a,a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
591                   'One or more values of the ',trim(curvar%p%name),' profile at ',yy,mm,dd,hh,min,ss, &
592                   ' exceed the prescribed maximum of ',curvar%p%maximum,'.'
593                call fatal_error('input::profile_file_update', trim(self%path)//': '//trim(strline))
594             end if
595             curvar => curvar%next
596          end do
597
598          self%nprofiles = self%nprofiles + 1
599          call julian_day(yy,mm,dd,self%jul2)
600          self%secs2 = hh*3600 + min*60 + ss
601          if(time_diff(self%jul2,self%secs2,jul,secs) > 0) exit
602       end do
603       if (self%nprofiles == 1) call fatal_error('input::profile_file_update', 'Simulation starts before time of first observation in '//trim(self%path)//'.')
604
605       ! Compute slopes (change in variable per second)
606       dt = time_diff(self%jul2,self%secs2,self%jul1,self%secs1)
607       self%alpha = (self%prof2-self%prof1)/dt
608    end if
609
610    ! Perform time interpolation
611    t = time_diff(jul,secs,self%jul1,self%secs1)
612    curvar => self%variables%first
613    do while (associated(curvar))
614       curvar%p%data = self%prof1(:,curvar%p%index) + t * self%alpha(:,curvar%p%index)
615       curvar => curvar%next
616    end do
617
618    end subroutine profile_file_update
619 !EOC
620
621 !-----------------------------------------------------------------------
622 !BOP
623 !
624 ! !IROUTINE: Initialize a single input file with horizontal (ØD) variables.
625 !
626 ! !INTERFACE:
627    subroutine timeseries_file_initialize(self)
628 !
629 ! !ÐESCRIPTION:
630 !  Initialize a single file with observed profiles.
631 !
632 ! !INPUT PARAMETERS:
633    class (type_timeseries_file),intent(inout) :: self
634 !
635 ! !REVISION HISTORY:
636 !  Original author(s): Jorn Bruggeman
637 !
638 !EOP
639 !
640 ! !LOCAL VARIABLES:
641    type (type_scalar_input_node),pointer :: curvar
642    integer :: nvar
643    integer :: rc
644    integer :: ios
645 !
646 !-----------------------------------------------------------------------
647 !BOC
648 !  Open the input file.
649    open(next_unit_no,file=self%path,status='old',action='read',iostat=ios)
650    if (ios /= 0) call fatal_error('input::timeseries_file_initialize', 'Unable to open "'//trim(self%path)//'" for reading')
651
652 !  Opening was successful - store the file unit, and increment the next unit with 1.
653    self%unit = next_unit_no
654    next_unit_no = next_unit_no + 1
655
656 !  Ðetermine the maximum number of columns that we need to read.
657    nvar = 0
658    curvar => self%variables%first
659    do while (associated(curvar))
660       nvar = max(nvar,curvar%p%index)
661       curvar => curvar%next
662    end do
663
664    allocate(self%obs1(nvar),stat=rc)
665    if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (obs1)'
666    self%obs1 = 0
667
668    allocate(self%obs2(nvar),stat=rc)
669    if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (obs2)'
670    self%obs2 = 0
671
672    allocate(self%alpha(nvar),stat=rc)
673    if (rc /= 0) stop 'input::timeseries_file_initialize: Error allocating memory (alpha)'
674    self%alpha = 0
675
```

```
676    end subroutine timeseries_file_initialize
677 !EOC
678
679 !-----------------------------------------------------------------------
680 !BOP
681 !
682 ! !IROUTINE: Read 0D data from a single input file
683 !
684 ! !INTERFACE:
685    subroutine timeseries_file_update(self,jul,secs)
686 !
687 ! !DESCRIPTION:
688 !  Get observations for the current time from a single input file.
689 !  This reads in new observations if necessary (and available),
690 !  and performs linear interpolation in time.
691 !
692 ! !USES:
693    use time, only: time_diff,julian_day
694 !
695 ! !INPUT PARAMETERS:
696    integer,                    intent(in)    :: jul,secs
697 !
698 ! !INPUT/OUTPUT PARAMETERS:
699    class(type_timeseries_file), intent(inout) :: self
700 !
701 ! !REVISION HISTORY:
702 !  Original author(s): Jorn Bruggeman
703 !
704 !EOP
705 !
706 ! !LOCAL VARIABLES:
707    integer                      :: rc
708    integer                      :: yy,mm,dd,hh,mins,ss
709    REALTYPE                     :: t,dt
710    type (type_scalar_input_node),pointer :: curvar
711    character(len=128)           :: strline
712 !
713 !-----------------------------------------------------------------------
714 !BOC
715    if (self%unit==-1) call self%initialize()
716
717 !  This part reads in new values if necessary.
718    if(time_diff(self%jul2,self%secs2,jul,secs) < 0) then
719       do
720          self%jul1 = self%jul2
721          self%secs1 = self%secs2
722          self%obs1 = self%obs2
723          call read_obs(self%unit,yy,mm,dd,hh,mins,ss,size(self%obs2),self%obs2,rc,line=self%lines)
724          if (rc>0) then
725             write (strline,'(i0)') self%lines
726             call fatal_error('input::timeseries_file_update', 'Error reading time series from '//trim(self%path)//' a
    t line '//strline)
727          elseif (rc<0) then
728             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?')
729          end if
730
731          ! Apply offsets and scale factors to newly read data
732          curvar => self%variables%first
733          do while (associated(curvar))
734             self%obs2(curvar%p%index) = curvar%p%scale_factor * self%obs2(curvar%p%index) + curvar%p%add_offset
735             if (self%obs2(curvar%p%index) < curvar%p%minimum) then
736                write (strline,'(a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
737                   trim(curvar%p%name),' at ',yy,mm,dd,hh,mins,ss, &
738                   ' lies below prescribed minimum of ',curvar%p%minimum,'.'
739                call fatal_error('input::timeseries_file_update', trim(self%path)//': '//trim(strline))
740             end if
741             if (self%obs2(curvar%p%index) > curvar%p%maximum) then
742                write (strline,'(a,a,i0.4,"-",i0.2,"-",i0.2," ",i0.2,":",i0.2,":",i0.2,a,g13.6,a)') &
743                   trim(curvar%p%name),' at ',yy,mm,dd,hh,mins,ss, &
744                   ' exceeds the prescribed maximum of ',curvar%p%maximum,'.'
745                call fatal_error('input::timeseries_file_update', trim(self%path)//': '//trim(strline))
746             end if
747             curvar => curvar%next
748          end do
749
750          self%n = self%n + 1
751          call julian_day(yy,mm,dd,self%jul2)
752          self%secs2 = hh*3600 + mins*60 + ss
753          if(time_diff(self%jul2,self%secs2,jul,secs) > 0) exit
754       end do
755       if (self%n == 1) call fatal_error('input::timeseries_file_update', 'Simulation starts before time of first obse
    rvation in '//trim(self%path)//'.')
756
757       ! Compute slopes (change in variable per second)
758       dt = time_diff(self%jul2,self%secs2,self%jul1,self%secs1)
759       self%alpha = (self%obs2 - self%obs1) / dt
760    end if
761
762    ! Perform time interpolation
763    t = time_diff(jul,secs,self%jul1,self%secs1)
764    curvar => self%variables%first
765    do while (associated(curvar))
766       curvar%p%value = min(max(self%obs1(curvar%p%index), self%obs2(curvar%p%index)), max(min(self%obs1(curvar%p%inde
    x), self%obs2(curvar%p%index)), self%obs1(curvar%p%index) + t * self%alpha(curvar%p%index)))
767       curvar => curvar%next
768    end do
769
```

```
   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
```

```
!
! !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
```

```
964          if (ios /= 0) return
965
966          if (len_trim(cbuf) > 0 .and. .not. (cbuf(1:1) == '#' .or. cbuf(1:1) == '!')) then
967             read(cbuf,*,iostat=ios) tmp_depth(i),(tmp_profs(i,j),j=1,cols)
968             if (ios < 0) ios = 1   ! End-of-file (ios<0) means premature end of line, which is a read error (ios>0) t
    o us
969             if (ios /= 0) return
970             exit
971          end if
972       end do
973    end do
974
975    call gridinterpol(N,cols,tmp_depth,tmp_profs,nlev,z,profiles)
976
977    deallocate(tmp_depth)
978    deallocate(tmp_profs)
979
980    end subroutine read_profiles
981 !EOC
982
983    subroutine fatal_error(location,error)
984       character(len=*),  intent(in) :: location,error
985
986       FATAL trim(location)//': '//trim(error)
987       stop 1
988    end subroutine fatal_error
989
990 !-----------------------------------------------------------------------
991
992    end module input
993
994 !-----------------------------------------------------------------------
995 ! Copyright by the GOTM-team under the GNU Public License - www.gnu.org
996 !-----------------------------------------------------------------------
997
```