```
#include "cppdefs.h"
!-----------------------------------------------------------------------
!BOP
!
! !MODULE:  time --- keep control of time \label{sec:time}
!
! !INTERFACE:
   MODULE time
!
! !DESCRIPTION:
!  This module provides a number of routines/functions and variables
!  related to the mode time in GOTM.
!  The basic concept used in this module is that time is expressed
!  as two integers --- one is the true Julian day and the other is
!  seconds since midnight. All calculations with time then become
!  very simple operations on integers.
!
! !USES:
   IMPLICIT NONE
!
!  default: all is private.
   private
!
! !PUBLIC MEMBER FUNCTIONS:
   public                              :: init_time, calendar_date
   public                              :: julian_day, update_time
   public                              :: read_time_string
   public                              :: write_time_string
   public                              :: time_diff
   public                              :: sunrise_sunset
   public                              :: in_time_interval
#ifdef _PRINTSTATE_
   public                              :: print_state_time
#endif
!
! !PUBLIC DATA MEMBERS:
   character(len=19), public           :: timestr
   character(len=19), public           :: start
   character(len=19), public           :: stop
   REALTYPE,          public           :: timestep
   REALTYPE,          public           :: fsecs,simtime,fsecondsofday
   integer,target,    public           :: julianday,secondsofday
   integer,target,    public           :: yearday
   integer,           public           :: timefmt
   integer,parameter, public           :: timestepkind = selected_int_kind(12)
   integer(kind=timestepkind), public  :: MinN,MaxN
   integer, public                     :: jul2,secs2
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
!
! !PRIVATE DATA MEMBERS:
   logical                   :: HasRealTime
   integer                   :: jul0,secs0
!
!-----------------------------------------------------------------------
!
   contains

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE: Initialise the time system
!
! !INTERFACE:
   subroutine init_time(MinN,MaxN)
!
! !DESCRIPTION:
!  The subroutine {\tt init\_time()} initialises the time module by reading
!  a namelist and take actions according to the specifications.
!  On exit from this subroutine the two variables MinN and MaxN have well
!  defined values and can be used in the time loop.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT/OUTPUT PARAMETERS:
   integer(kind=timestepkind), intent(inout)    :: MinN,MaxN
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
!
! !LOCAL VARIABLES:
   integer                     :: jul1,secs1
   integer(kind=timestepkind) :: nsecs
   integer                     :: ndays
!
!-----------------------------------------------------------------------
!BOC
!  Read time specific things from the namelist.
!
   LEVEL1 'init_time'
!
!  Calculate MaxN -> MinN is 1 if not changed by HotStart
```

```
!
   MinN = 1
   LEVEL2 'Time step:      ',timestep,' seconds'
   LEVEL2 'Time format:    ',timefmt
   select case (timefmt)
      case (1)
         HasRealTime=.false.
         LEVEL2 '# of timesteps: ',MaxN
         start='2000-01-01 00:00:00'
         call read_time_string(start,jul1,secs1)
         LEVEL2 'Fake start:     ',start
      case (2)
         HasRealTime=.true.
         LEVEL2 'Start:          ',start
         LEVEL2 'Stop:           ',stop
         call read_time_string(start,jul1,secs1)
         call read_time_string(stop,jul2,secs2)

         nsecs = time_diff(jul2,secs2,jul1,secs1)
         MaxN  = nint(nsecs/timestep,kind=timestepkind)

         ndays = nsecs/86400
         nsecs = nsecs - 86400*ndays
         STDERR '        ==> ',ndays,' day(s) and ',nsecs,' seconds ==> ',MaxN,' micro time steps'
      case (3)
         HasRealTime=.true.
         LEVEL2 'Start:          ',start
         LEVEL2 '# of timesteps: ',MaxN

         call read_time_string(start,jul1,secs1)

         nsecs = nint(MaxN*timestep,kind=timestepkind) + secs1
         ndays = nsecs/86400
         jul2  = jul1 + ndays
         secs2 = mod(nsecs,86400_timestepkind)

         call write_time_string(jul2,secs2,stop)
         LEVEL2 'Stop:           ',stop
      case default
         STDERR 'Fatal error: A non valid input format has been chosen'
         stop 'init_time'
   end select

   jul0  = jul1
   secs0 = secs1

   call update_time(0_timestepkind)

   simtime = timestep*(MaxN-MinN+1)

   return
   end subroutine init_time
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE:  Convert true Julian day to calendar date
!
! !INTERFACE:
   subroutine calendar_date(julian,yyyy,mm,dd)
!
! !DESCRIPTION:
!  Converts a Julian day to a calendar date --- year, month and day.
!  Based on a similar routine in \emph{Numerical Recipes}.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   integer                             :: julian
!
! !OUTPUT PARAMETERS:
   integer                             :: yyyy,mm,dd
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
!
! !LOCAL VARIABLES:
   integer, parameter        :: IGREG=2299161
   integer                   :: ja,jb,jc,jd,je
   REAL                      :: x
!
!-----------------------------------------------------------------------
!BOC
   if(julian .ge. IGREG ) then
      x = ((julian-1867216)-0.25)/36524.25
      ja = julian+1+int(x)-int(0.25*x)
   else
      ja = julian
   end if

   jb = ja+1524
   jc = int(6680 + ((jb-2439870)-122.1)/365.25)
   jd = int(365*jc+(0.25*jc))
   je = int((jb-jd)/30.6001)
```

```
197
198    dd = jb-jd-int(30.6001*je)
199    mm = je-1
200    if (mm .gt. 12) mm = mm-12
201    yyyy = jc - 4715
202    if (mm .gt. 2) yyyy = yyyy-1
203    if (yyyy .le. 0) yyyy = yyyy-1
204
205    return
206    end subroutine calendar_date
207 !EOC
208
209 !-----------------------------------------------------------------------
210 !BOP
211 !
212 ! !IROUTINE:  Convert a calendar date to true Julian day
213 !
214 ! !INTERFACE:
215    subroutine julian_day(yyyy,mm,dd,julian)
216 !
217 ! !ÐESCRIPTION:
218 !  Converts a calendar date to a Julian day.
219 !  Based on a similar routine in \emph{Numerical Recipes}.
220 !
221 ! !USES:
222    IMPLICIT NONE
223 !
224 ! !INPUT PARAMETERS:
225    integer                              :: yyyy,mm,dd
226 !
227 ! !OUTPUT PARAMETERS:
228    integer                              :: julian
229 !
230 ! !REVISION HISTORY:
231 !  Original author(s): Karsten Bolding & Hans Burchard
232 !
233 !EOP
234 !
235 ! !LOCAL VARIABLES:
236    integer, PARAMETER        :: IGREG=15+31*(10+12*1582)
237    integer                   :: ja,jy,jm
238 !
239 !-----------------------------------------------------------------------
240 !BOC
241    jy = yyyy
242    if(jy .lt. 0) jy = jy+1
243    if (mm .gt. 2) then
244       jm = mm+1
245    else
246       jy = jy-1
247       jm = mm+13
248    end if
249    julian = int(floor(365.25*jy)+floor(30.6001*jm)+dd+1720995)
250    if (dd+31*(mm+12*yyyy) .ge. IGREG) then
251       ja = int(0.01*jy)
252       julian = julian+2-ja+int(0.25*ja)
253    end if
254
255    return
256    end subroutine julian_day
257 !EOC
258
259 !-----------------------------------------------------------------------
260 !BOP
261 !
262 ! !IROUTINE:  Keep track of time (Julian days and seconds)
263 !
264 ! !INTERFACE:
265    subroutine update_time(n)
266 !
267 ! !ÐESCRIPTION:
268 !  Based on a starting time this routine calculates the actual time
269 !  in a model integration using the number of time steps, {\tt n},
270 !  and the size of the time step, {\tt timestep}. More public variables
271 !  can be updated here if necessary.
272 !
273 ! !USES:
274    IMPLICIT NONE
275 !
276 ! !INPUT PARAMETERS:
277    integer(kind=timestepkind), intent(in) :: n
278 !
279 ! !REVISION HISTORY:
280 !  Original author(s): Karsten Bolding & Hans Burchard
281 !
282 !EOP
283 !
284 ! !LOCAL VARIABLES:
285    integer(kind=timestepkind) :: nsecs
286    integer                    :: yyyy,mm,dd,jd_firstjan
287 !
288 !-----------------------------------------------------------------------
289 !BOC
290    nsecs = nint(n*timestep,kind=timestepkind) + secs0
291    fsecs = n*timestep + secs0
292    julianday    = jul0 + nsecs/86400
293    secondsofday = mod(nsecs,86400_timestepkind)
294    fsecondsofday = mod(fsecs,real(86400,kind(_ONE_)))
```

```
295
296    call calendar_date(julianday,yyyy,mm,dd)
297    call julian_day(yyyy,1,1,jd_firstjan)
298    yearday = julianday-jd_firstjan+1
299
300    return
301    end subroutine update_time
302 !EOC
303
304 !-----------------------------------------------------------------------
305 !BOP
306 !
307 ! !IROUTINE:  Convert a time string to Julian day and seconds
308 !
309 ! !INTERFACE:
310    subroutine read_time_string(timestr,jul,secs)
311 !
312 ! !DESCRIPTION:
313 !  Converts a time string to the true Julian day and seconds of that day.
314 !  The format of the time string must be: {\tt yyyy-mm-dd hh:hh:ss }.
315 !
316 ! !USES:
317    IMPLICIT NONE
318 !
319 ! !INPUT PARAMETERS:
320    character(len=19)                   :: timestr
321 !
322 ! !OUTPUT PARAMETERS:
323    integer, intent(out)                :: jul,secs
324 !
325 ! !REVISION HISTORY:
326 !  Original author(s): Karsten Bolding & Hans Burchard
327 !
328 !EOP
329 !
330 ! !LOCAL VARIABLES:
331    character                 :: c1,c2,c3,c4
332    integer                   :: yy,mm,dd,hh,min,ss
333 !
334 !-----------------------------------------------------------------------
335 !BOC
336    read(timestr,'(i4,a1,i2,a1,i2,1x,i2,a1,i2,a1,i2)')  &
337                           yy,c1,mm,c2,dd,hh,c3,min,c4,ss
338    call julian_day(yy,mm,dd,jul)
339    secs = 3600*hh + 60*min + ss
340
341    return
342    end subroutine read_time_string
343 !EOC
344
345 !-----------------------------------------------------------------------
346 !BOP
347 !
348 ! !IROUTINE:  Convert Julian day and seconds into a time string
349 !
350 ! !INTERFACE:
351    subroutine write_time_string(jul,secs,timestr)
352 !
353 ! !DESCRIPTION:
354 !  Formats Julian day and seconds of that day to a nice looking
355 !  character string.
356 !
357 ! !USES:
358    IMPLICIT NONE
359 !
360 ! !INPUT PARAMETERS:
361    integer, intent(in)                 :: jul,secs
362 !
363 ! !OUTPUT PARAMETERS:
364    character(len=19)                   :: timestr
365 !
366 ! !REVISION HISTORY:
367 !  Original author(s): Karsten Bolding & Hans Burchard
368 !
369 !EOP
370 !
371 ! !LOCAL VARIABLES:
372    integer                   :: ss,min,hh,dd,mm,yy
373 !
374 !-----------------------------------------------------------------------
375 !BOC
376    hh   = secs/3600
377    min  = (secs-hh*3600)/60
378    ss   = secs - 3600*hh - 60*min
379
380    call calendar_date(jul,yy,mm,dd)
381
382    write(timestr,'(i4.4,a1,i2.2,a1,i2.2,1x,i2.2,a1,i2.2,a1,i2.2)')  &
383                          yy,'-',mm,'-',dd,hh,':',min,':',ss
384
385    return
386    end subroutine write_time_string
387 !EOC
388
389 !-----------------------------------------------------------------------
390 !BOP
391 !
392 ! !IROUTINE:  Return the time difference in seconds
```

```
!
! !INTERFACE:
   REALTYPE FUNCTION time_diff(jul1,secs1,jul2,secs2)
!
! !ÐESCRIPTION:
! This functions returns the time difference between two
! dates in seconds. The dates are given as Julian day and seconds
! of that day.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   integer, intent(in)                :: jul1,secs1,jul2,secs2
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
!EOP
!-----------------------------------------------------------------------
!BOC
   time_diff = 86400.0d0 *(jul1-jul2) + _ONE_*(secs1-secs2)

   return
   end function  time_diff
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE:  Return the times of sunrise and sunset
!
! !INTERFACE:
   subroutine sunrise_sunset(latitude,declination,sunrise,sunset)
!
! !ÐESCRIPTION:
! This functions returns the time difference between two
! dates in seconds. The dates are given as Julian day and seconds
! of that day.
!
! !USES:
   IMPLICIT NONE
!
! !INPUT PARAMETERS:
   REALTYPE, intent(in)               :: latitude,declination
!
! !OUTPUT PARAMETERS:
   REALTYPE, intent(out)              :: sunrise,sunset
!
! !REVISION HISTORY:
!  Original author(s): Karsten Bolding & Hans Burchard
!
! !LOCAL VARIABLES:
   REALTYPE                  :: omega,hour
!EOP
!-----------------------------------------------------------------------
!BOC
   omega = acos(-tan(latitude*3.141516/180.)*tan(declination*3.141516/180.))
   hour  = omega*180/3.141516/15.
   sunrise = 12. - hour
   sunset  = 12. + hour
   return
   end subroutine  sunrise_sunset
!EOC

!-----------------------------------------------------------------------
!BOP
!
! !IROUTINE:  in_interval() -
!
! !INTERFACE:
   logical function in_time_interval(j1,s1,j,s,j2,s2)
!
! !ÐESCRIPTION:
!
! !USES:
!
! !INPUT PARAMETERS:
   integer, intent(in)                :: j1,s1,j,s,j2,s2
!
! !REVISION HISTORY:
!  22Nov Author name Initial code
!
! !LOCAL VARIABLES:
   logical         :: before,after
!EOP
!-----------------------------------------------------------------------
!BOC

   before = (j .lt. j1) .or. ( j .eq. j1 .and. (s .lt. s1) )
   after  = (j .gt. j2) .or. ( j .eq. j2 .and. (s .gt. s2) )

   in_time_interval = ( .not. before ) .and. ( .not. after )
   return
   end function in_time_interval
!EOC

```

```
491 #ifdef _PRINTSTATE_
492 !-------------------------------------------------------------------------
493 !BOP
494 !
495 ! !IROUTINE: Print the current state of the time module.
496 !
497 ! !INTERFACE:
498    subroutine print_state_time()
499 !
500 ! !DESCRIPTION:
501 !  This routine writes the value of all module-level variables to screen.
502 !
503 ! !USES:
504    IMPLICIT NONE
505 !
506 ! !REVISION HISTORY:
507 !  Original author(s): Jorn Bruggeman
508 !
509 !EOP
510 !-------------------------------------------------------------------------
511 !BOC
512    LEVEL1 'State of time module:'
513    LEVEL2 'timestr',timestr
514    LEVEL2 'start',start
515    LEVEL2 'stop',stop
516    LEVEL2 'timestep',timestep
517    LEVEL2 'fsecs,simtime',fsecs,simtime
518    LEVEL2 'julianday,secondsofday',julianday,secondsofday
519    LEVEL2 'yearday',yearday
520    LEVEL2 'timefmt',timefmt
521    LEVEL2 'MinN,MaxN',MinN,MaxN
522    LEVEL2 'HasRealTime',HasRealTime
523    LEVEL2 'jul0,secs0',jul0,secs0
524
525    end subroutine print_state_time
526 !EOC
527 #endif
528
529 !-------------------------------------------------------------------------
530
531    end module time
532
533 !-------------------------------------------------------------------------
534 ! Copyright by the GOTM-team under the GNU Public License - www.gnu.org
535 !-------------------------------------------------------------------------
```